

Chemical Engineering Computation with MATLAB®



Yeong Koo Yeo



CRC Press
Taylor & Francis Group

Chemical Engineering Computation with MATLAB®



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Chemical Engineering Computation with MATLAB®

Yeong Koo Yeo



CRC Press

Taylor & Francis Group

Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

MATLAB® is a trademark of The MathWorks, Inc. and is used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book's use or discussion of MATLAB® software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or particular use of the MATLAB® software.

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Taylor & Francis Group
Boca Raton, FL 33487-2742

© 2018 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed on acid-free paper

International Standard Book Number-13: 978-1-138-03989-6 (Hardback)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Library of Congress Cataloging-in-Publication Data

Names: Yeo, Yeong Koo, author.
Title: Chemical engineering computation with MATLAB / Yeong Koo Yeo.
Description: Boca Raton : Taylor & Francis, CRC Press, 2017. | Includes bibliographical references.
Identifiers: LCCN 2017011618 | ISBN 9781138039896 (hardback : acid-free paper)
| ISBN 9781315114880 (ebook)
Subjects: LCSH: Chemical engineering--Data processing. | MATLAB.
Classification: LCC TP184 .Y46 2017 | DDC 660.0285--dc23
LC record available at <https://lccn.loc.gov/2017011618>

Visit the Taylor & Francis Web site at
<http://www.taylorandfrancis.com>

and the CRC Press Web site at
<http://www.crcpress.com>

Contents

Preface.....	xv
Acknowledgments.....	xvii
Author	xix

Chapter 1 Introduction to MATLAB®	1
1.1 Starting MATLAB®	1
1.1.1 Entering Commands in the Command Window	1
1.1.2 Help Command	3
1.1.3 Exit from MATLAB®	4
1.2 Operations and Assignment of Variables.....	4
1.2.1 Errors in Input	5
1.2.2 Aborting Calculations	6
1.3 Vectors and Matrices	6
1.3.1 Vectors	6
1.3.2 Matrices	9
1.3.3 Complex Number	13
1.3.4 Suppression of Screen Output	14
1.4 Numerical Expressions	14
1.5 Managing Variables	16
1.5.1 <code>clear</code> Command	16
1.5.2 Computational Limitations and Constants	16
1.5.3 <code>whos</code> Command.....	17
1.6 Symbolic Operations	17
1.6.1 Substitution in Symbolic Equations	18
1.7 M-Files	19
1.7.1 Script M-Files.....	19
1.7.2 Adding Comments.....	20
1.7.3 Function M-Files	21
1.8 Functions	22
1.8.1 Built-In Functions	22
1.8.2 User-Defined Functions.....	23
1.9 Loops	24
1.9.1 <code>if</code> Statement	24
1.9.2 <code>for</code> Loop	25
1.10 Graphics.....	25
1.10.1 Plotting with <code>ezplot</code>	25
1.10.2 Modifying Graphs	25
1.10.3 Graphing with <code>plot</code>	27
1.10.4 Plotting Multiple Curves	28
1.10.5 Three-Dimensional Plots.....	29
1.11 Ordinary Differential Equations	30
1.12 Examples	31
1.12.1 Population Growth Model	31
1.12.2 Random Fibonacci Sequence	32
1.12.3 Generation of a 3D Object.....	33
Bibliography.....	34

Chapter 2	Numerical Methods with MATLAB®	35
2.1	Linear Systems	35
2.2	Nonlinear Equations	40
2.2.1	Polynomial Equations	40
2.2.2	Zeros of Nonlinear Equations.....	41
2.2.2.1	Bisection Method.....	41
2.2.2.2	False Position Method	41
2.2.2.3	Newton's Method.....	41
2.2.2.4	Secant Method	41
2.2.2.5	Muller's Method	41
2.2.2.6	Newton's Method for Nonlinear Systems.....	42
2.2.2.7	Fixed-Point Iteration Method for Nonlinear Systems.....	42
2.3	Regression Analysis.....	47
2.3.1	Introduction to Statistics.....	47
2.3.2	Generation of Random Numbers.....	50
2.3.3	Linear Regression Analysis	50
2.3.4	Polynomial Regression Analysis	52
2.3.5	Transformation of Nonlinear Functions to Linear Functions.....	53
2.4	Interpolation	60
2.4.1	Polynomial Interpolation.....	60
2.4.2	Use of the Function <i>polyfit</i>	62
2.4.3	Cubic Spline Interpolation	64
2.4.4	Interpolation in One Dimension	65
2.4.5	Interpolation in Multidimension.....	67
2.5	Optimization	69
2.6	Differentiation and Integration	70
2.6.1	Differentiation	70
2.6.2	Integration	72
2.6.2.1	Definite Integral	72
2.6.2.2	Multiple Integrals.....	74
2.7	Ordinary Differential Equations	75
2.7.1	Initial-Value Problems.....	75
2.7.2	Stiff System	77
2.7.3	Boundary-Value Problems.....	84
2.8	Partial Differential Equations	87
2.8.1	Classification of Partial Differential Equations	87
2.8.1.1	Classification by Order	87
2.8.1.2	Classification by Linearity	87
2.8.1.3	Classification of Linear 2nd-Order Partial Differential Equations	88
2.8.1.4	Classification by Initial and Boundary Conditions	88
2.8.2	Solution of 1st-Order Parabolic/Elliptic Partial Differential Equations Using <i>pdepe</i>	88
2.9	Historical Development of Process Engineering Software	91
	Problems.....	93
	References	101
Chapter 3	Physical Properties	103
3.1	Water and Steam	103
3.1.1	Division of Pressure–Temperature Range	103

3.1.2	Property Equations	104
3.1.2.1	Parameters and Auxiliary Equations	104
3.1.2.2	Basic Equations for Region 1	104
3.1.2.3	Basic Equations for Region 2	105
3.1.2.4	Basic Equations for Region 3	105
3.1.2.5	Basic Equations for Region 4	105
3.1.2.6	Basic Equations for Region 5	106
3.1.3	Properties of Saturated Steam	110
3.2	Humidity	113
3.2.1	Relative Humidity	113
3.2.2	Absolute Humidity	114
3.3	Density of Saturated Liquids	115
3.3.1	COSTALD Method	117
3.4	Viscosity	118
3.4.1	Viscosity of Liquids	118
3.4.2	Viscosity of Gases	120
3.5	Heat Capacity	121
3.5.1	Heat Capacity of Liquids	121
3.5.1.1	Polynomial Correlation	121
3.5.1.2	Rowlinson/Bondi Method	123
3.5.2	Heat Capacities of Gases	123
3.6	Thermal Conductivity	125
3.6.1	Thermal Conductivities of Liquids	125
3.6.2	Thermal Conductivities of Gases	126
3.7	Surface Tension	128
3.8	Vapor Pressure	132
3.8.1	Antoine Equation	132
3.8.1.1	Extended Antoine Equation	132
3.8.2	Wagner Equation	134
3.8.3	Hoffmann–Florin Equation	135
3.8.4	Rarey/Moller Equation	135
3.9	Enthalpy of Vaporization	137
3.9.1	Watson Equation	137
3.9.2	Pitzer Correlation	138
3.9.3	Clausius–Clapeyron Equation	139
3.10	Heat of Formation for Ideal Gases	140
3.11	Gibbs Free Energy	142
3.12	Diffusion Coefficients	143
3.12.1	Liquid-Phase Diffusion Coefficients	143
3.12.2	Gas-Phase Diffusion Coefficients	144
3.13	Compressibility Factor of Natural Gases	145
	Problems	147
	References	148
Chapter 4	Thermodynamics	151
4.1	Equation of State	151
4.1.1	Virial State Equation	151
4.1.2	Lee–Kesler Equation	153
4.1.3	Cubic Equations of State	155

4.2	Thermodynamic Properties of Fluids	158
4.2.1	Enthalpy Change	158
4.2.2	Departure Function.....	158
4.2.2.1	Departure Function from the Virial Equation of State	159
4.2.2.2	Departure Function from the VDW (van der Waals) Equation of State.....	159
4.2.2.3	Departure Function from the RK (Redlich–Kwong) Equation of State.....	159
4.2.2.4	Departure Function from the SRK (Soave–Redlich–Kwong) Equation of State	160
4.2.2.5	Departure Function from the PR (Peng–Robinson) Equation of State.....	160
4.2.3	Enthalpy of Mixture	164
4.3	Fugacity Coefficient	168
4.3.1	Fugacity Coefficients of Pure Species	168
4.3.1.1	Fugacity Coefficient from the Virial Equation of State	168
4.3.1.2	Fugacity Coefficient from the van der Waals Equation of State	168
4.3.1.3	Fugacity Coefficient from the Redlich–Kwong Equation of State.....	168
4.3.1.4	Fugacity Coefficient from the Soave–Redlich–Kwong and Peng–Robinson Equations of State	168
4.3.1.5	Fugacity Coefficient from Experimental Data.....	169
4.3.2	Fugacity Coefficient of a Species in a Mixture	171
4.3.2.1	Fugacity Coefficient from the Virial Equation of State	171
4.3.2.2	Fugacity Coefficient from the Cubic Equations of State	171
4.3.2.3	Fugacity Coefficient from the van der Waals Equation of State	171
4.4	Vapor Pressure	173
4.4.1	Antoine Equation.....	173
4.4.2	Riedel Equation	174
4.4.3	Harlecher–Braun Equation	174
4.5	Activity Coefficient	176
4.5.1	Activity Coefficient Models	179
4.5.1.1	Wilson Equation.....	179
4.5.2	van Laar Equation	179
4.5.3	Activity Coefficients by the Group Contribution Method	180
4.5.3.1	UNIQUAC Method.....	180
4.5.3.2	UNIFAC Method	181
4.6	Vapor–Liquid Equilibrium.....	185
4.6.1	Vapor–Liquid Equilibrium by Raoult’s Law	185
4.6.2	Vapor–Liquid Equilibrium by Modified Raoult’s Law	188
4.6.2.1	Flash Calculations.....	192
4.6.3	Vapor–Liquid Equilibrium Using Ratio of Fugacity Coefficients	195
4.6.3.1	Dew-Point and Bubble-Point Calculations	195
4.6.3.2	Flash Calculations.....	199
4.7	Vapor–Liquid–Liquid Equilibrium.....	203
4.7.1	Bubble-Point Calculations.....	204

4.7.2	Dew-Point Calculations.....	204
4.7.3	Isothermal Flash Calculations in the Two-Phase Region	205
Problems.....		207
References		210
Chapter 5	Fluid Mechanics	213
5.1	Laminar Flow	213
5.1.1	Reynolds Number.....	213
5.1.2	Flow in a Horizontal Pipe.....	214
5.1.3	Laminar Flow in a Horizontal Annulus.....	215
5.1.4	Vertical Laminar Flow of a Falling Film.....	217
5.1.5	Falling Particles	219
5.2	Friction Factor	221
5.3	Flow of Fluids in Pipes	225
5.3.1	Friction Loss.....	225
5.3.2	Equivalent Length of Various Fittings and Valves.....	231
5.3.3	Excess Head Loss.....	231
5.3.4	Pipe Reduction and Enlargement	232
5.3.5	Overall Pressure Drop	234
5.3.6	Pipeline Network.....	237
5.4	Flow through Tank.....	242
5.4.1	Open Tank	242
5.4.2	Enclosed Tank	243
5.5	Flow of Non-Newtonian Fluid.....	245
5.5.1	Velocity Profile	245
5.5.2	Reynolds Number.....	247
5.5.3	Friction Factor	248
5.6	Compressible Fluid Flow in Pipes	250
5.6.1	Critical Flow and the Mach Number	250
5.6.2	Compressible Isothermal Flow.....	250
5.7	Two-Phase Flow in Pipes.....	254
5.7.1	Flow Patterns	254
5.7.2	Pressure Drop	255
5.7.3	Corrosion and Erosion.....	257
5.7.4	Vapor–Liquid Two-Phase Vertical Downflow	260
5.7.5	Pressure Drop in Flashing Steam Condensate Flow	260
5.8	Flow through Packed Beds	261
Problems.....		263
References		265
Chapter 6	Chemical Reaction Engineering.....	269
6.1	Reaction Rates	269
6.1.1	Reaction Rate Constant	269
6.1.1.1	Estimation of Reaction Order and Rate Constant Using Experimental Data.....	270
6.1.2	Reaction Equilibrium	272
6.1.3	Reaction Conversion	273
6.1.3.1	Estimation of Reaction Rate Parameters Using Conversion	274
6.1.4	Series Reactions	275

6.2	Continuous-Stirred Tank Reactor (CSTR).....	277
6.2.1	Concentration Changes with Time	277
6.2.2	Exothermic Reaction	278
6.2.3	Multiple Reactions in a CSTR	282
6.3	Batch Reactor	284
6.3.1	Estimation of Batch Reaction Parameters	284
6.3.2	Semibatch Reactors	285
6.4	Plug-Flow Reactor.....	287
6.5	Catalytic Reactors.....	294
6.5.1	Straight-Through Transport Reactor	302
6.6	Membrane Reactors.....	304
6.7	Bioreactors.....	308
6.8	Nonisothermal Reactions.....	311
6.8.1	Adiabatic Reaction	311
6.8.2	Steady-State Nonisothermal Reactions	314
	Problems	319
	References	327

Chapter 7	Mass Transfer	329
7.1	Diffusion	329
7.1.1	One-Dimensional Diffusion	329
7.1.2	Multicomponent Diffusion in Gases	330
7.1.3	Diffusion from a Sphere	333
7.1.4	Mass Transfer Coefficient	334
7.1.5	Diffusion in Isothermal Catalyst Particles.....	337
7.2	Unsteady-State Mass Transfer	343
7.2.1	Unsteady-State Diffusion in a One-Dimensional Slab.....	343
7.2.2	Diffusion in a Falling Laminar Film	345
7.3	Evaporation.....	348
7.3.1	Single-Effect Evaporator	348
7.3.2	Multiple-Effect Evaporators	351
7.4	Absorption	356
7.5	Binary Distillation	360
7.5.1	McCabe-Thiele Method	360
7.5.2	Ideal Binary Distillation	363
7.6	Shortcut Calculation Method for Multicomponent Distillation	367
7.6.1	Fenske–Underwood–Gilliland Method	367
7.6.1.1	Fenske Equation: The Minimum Number of Stages	367
7.6.1.2	Underwood Equation: The Minimum Reflux	368
7.6.1.3	Gilliland Correlation.....	368
7.6.1.4	Feed Stage Location	370
7.6.1.5	Determination of the Number of Stages by the Smoker Equation.....	370
7.7	Rigorous Steady-State Distillation Calculations	373
7.7.1	Rigorous Distillation Model: MESH Equations.....	376
7.7.2	Tridiagonal Matrix Method	379
7.7.3	Bubble-Point (BP) Method.....	382
7.8	Differential Distillation.....	390

7.9	Membrane Separation.....	394
7.9.1	Complete-Mixing Model for Gas Separation.....	395
7.9.2	Cross-Flow Model for Gas Separation	397
	Problems.....	400
	References	404
Chapter 8	Heat Transfer	407
8.1	One-Dimensional Heat Transfer.....	407
8.1.1	Heat Transfer in a One-Dimensional Slab.....	407
8.1.2	Heat Transfer through a Multilayer Slab.....	409
8.1.3	Heat Transfer through Multilayer Cylinders	410
8.1.4	Heat Transfer in a Wire	413
8.1.5	Heat Loss through Pipe Flange	415
8.2	Multidimensional Heat Conduction	416
8.2.1	Unsteady-State Heat Conduction	416
8.2.2	Method of Lines	418
8.2.3	Steady-State Heat Conduction.....	424
8.3	Heat Exchanger	429
8.3.1	Shell-and-Tube Heat Exchanger without Phase Change.....	429
8.3.1.1	Log-Mean Temperature Difference	429
8.3.2	Tube-Side Heat Transfer Coefficient.....	432
8.3.3	Shell-Side Heat Transfer Coefficient.....	433
8.3.4	Pressure Drop in the Tube Side	435
8.3.5	Pressure Drop in the Shell Side	435
8.3.6	Heat Exchanger Calculation Procedure.....	436
8.3.7	Double-Pipe Heat Exchanger	447
8.4	Heat Tracing in Pipelines.....	450
8.5	Air Cooler	453
	Problems.....	455
	References	457
Chapter 9	Process Control	459
9.1	Laplace Transform and Transfer Function	459
9.1.1	Laplace Transform and Inverse Laplace Transform.....	459
9.1.2	Partial Fraction Expansion	460
9.1.3	Representation of Transfer Function	461
9.2	Block Diagram.....	462
9.3	State-Space Representation	463
9.4	Process Dynamics.....	464
9.4.1	Dynamics of a 1st-Order Process	464
9.4.2	Dynamics of 2nd-Order Processes	468
9.4.3	Dynamics of Complex Processes	471
9.4.3.1	Higher-Order Processes	471
9.4.3.2	Lead/Lag	472
9.4.3.3	Time Delay	472
9.5	Dynamics of Feedback Control Loops	474
9.5.1	Closed-Loop Transfer Function	474
9.5.1.1	Servo Problem.....	474
9.5.1.2	Regulator Problem	475
9.5.2	Control of the Continuous Stirred Tank Heater.....	480

9.6	Stability of Feedback Control Systems	484
9.7	Frequency Response Analysis	487
9.7.1	Bode Diagram.....	487
9.7.2	Nyquist Diagram	487
9.7.3	Nichols Chart.....	487
9.7.4	Gain and Phase Margins.....	492
	Problems.....	493
Chapter 10	Optimization.....	497
10.1	Unconstrained Optimization.....	497
10.1.1	Fibonacci Method.....	497
10.1.2	Golden Section Method.....	499
10.1.3	Brent's Quadratic Fit Method.....	502
10.1.4	Shubert–Piyavskii Method	505
10.1.5	Steepest Descent Method	507
10.1.6	Newton's Method	510
10.1.7	Conjugate Gradient Method	512
10.1.8	Quasi-Newton Method	516
10.2	Linear Programming.....	520
10.2.1	Formulation of Linear Programming Problem.....	520
10.2.2	Simplex Method	521
10.2.3	Two-Phase Simplex Method.....	522
10.2.4	Interior Point Method	526
10.3	Constrained Optimization.....	528
10.3.1	Rosen's Gradient Projection Method	528
10.3.2	Zoutendijk's Feasible Direction Method.....	532
10.3.3	Generalized Reduced Gradient (GRG) Method	537
10.3.4	Sequential Quadratic Programming (SQP) Method.....	544
10.4	Direct Search Methods	549
10.4.1	Cyclic Coordinate Method	549
10.4.2	Hooke and Jeeves Pattern Search Method.....	551
10.4.3	Rosenbrock's Method.....	554
10.4.4	Nelder and Mead's Simplex Method	556
10.4.5	Simulated Annealing (SA) Method	559
10.4.6	Genetic Algorithm (GA).....	562
10.5	Mixed-Integer Programming	566
10.5.1	Zero–One Programming Method	566
10.5.2	Branch and Bound Method.....	569
10.6	Use of MATLAB Built-In Functions.....	575
10.6.1	Unconstrained Optimization.....	575
10.6.1.1	fminsearch.....	575
10.6.1.2	fminunc	575
10.6.1.3	fminbnd.....	576
10.6.1.4	lsqnonlin	576
10.6.2	Constrained Optimization	577
10.6.2.1	lsqlin	577
10.6.2.2	fmincon	577
10.6.2.3	fminimax	577
10.6.2.4	quadprog	578

10.6.3 Linear and Mixed-Integer Programming Problems.....	579
10.6.3.1 linprog.....	579
10.6.3.2 intlinprog.....	580
Problems.....	581
References	585
Appendix A: Supplementary Programs	587
Appendix B: List of Programs	593
Index.....	599



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Preface

OVERVIEW

This book focuses on the presentation of problem-solving techniques in the main subject areas of chemical engineering. Basics to advanced levels of problem solving are covered utilizing MATLAB®, which is a high-performance language for technical computing.

This book provides numerous examples and exercises, as well as extensive problem-solving instruction and solutions, for various chemical engineering problems. Solutions are developed using fundamental principles to construct mathematical models, and an equation-oriented approach is used to generate numerical results. For efficient problem solving, various numerical methods—including linear and nonlinear algebraic equations, finite difference methods, ordinary differential equations, boundary value problems, partial differential equations, curve fitting, and linear and nonlinear regressions—are employed in the examples.

BACKGROUND

Most problems encountered in chemical engineering are very sophisticated and interdisciplinary. Thus, calculations for chemical engineering problem solving by portable calculators or hand calculations are limited only to a few simple applications and problems. Since the introduction of personal computers and mathematical software packages in the early 1980s, emphasis has gradually shifted to computer-based problem solving. And it is increasingly becoming important for today's engineering students, researchers, and professionals to be proficient in the use of software tools for problem solving.

MATLAB is one such tool that has been found suitable for implementing algorithms required in engineering education, graduate research, advanced mathematics, and industrial operation-aid systems. It is distinguished by its ability to perform calculations in a vector-matrix form, a large library of built-in functions, strong structural language, and a rich set of graphical visualization tools. Furthermore, MATLAB integrates computations, visualization, and programming in an intuitive, user-friendly environment.

These useful features of MATLAB can be effectively applied in chemical engineering problem-solving activities. Examples and problems introduced in this book demonstrate how some of these special capabilities of MATLAB can be best used for effective and efficient problem solving.

FEATURES OF THE BOOK

This book is designed for chemical engineering students and industrial professionals. MATLAB is adopted as the computation environment throughout this book. This book provides examples and problems extracted from core chemical engineering subject areas and presents a basic instruction in the use of MATLAB for problem solving. Emphasis is placed on setting up problems systematically and obtaining required solutions efficiently. Additionally, since this book presents examples and problems commonly encountered in most chemical engineering subject areas, it can serve as a reference in various scenarios and environments.

Since all examples and problems presented in this book are solved with MATLAB, this book can also be a resource in developing proficiency in the use and application of MATLAB. Practicing engineers and students alike can learn to write MATLAB programs for various chemical engineering applications. (It should be noted that all programs provided in this book are written in MATLAB 2015a—but are also compatible with MATLAB 2014a/b or lower.)

Examples presented in this book demonstrate the implementation of various problem-solving approaches and methodologies for problem formulation, problem solving, analysis, and presentation—as well as visualization and documentation of results. This book also provides aid with advanced problems that are often encountered in graduate research and industrial operations, such as nonlinear regression, parameter estimation in differential systems, two-point boundary value problems, and partial differential equations and optimization.

INTENDED READERS

This book is intended for those with interest in learning how to use MATLAB to solve chemical engineering problems using computers. This book can be used as a textbook in a one-semester course for students in chemical engineering and related disciplines. For undergraduate students, this book can be a resource for learning how to classify and analyze problems according to the numerical methods that facilitate efficient and effective computations. This book can also be utilized as a reference for chemical engineering researchers and engineers, particularly in computer-aided problem solving.

Yeong Koo Yeo, PhD

MATLAB® is a registered trademark of The MathWorks, Inc. For product information, please contact:

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098 USA
Tel: 508-647-7000
Fax: 508-647-7001
E-mail: info@mathworks.com
Web: www.mathworks.com

Acknowledgments

This book is a result of my research and teaching career in chemical engineering, and I gratefully acknowledge the contributions of my students to the development of many of the ideas contained herein.

I especially thank the late Professor Chang Kyun Choi at Seoul National University, whose passion, knowledge, and discipline in this field have been a tremendous inspiration for my work.

I am indebted to Professor Kyu Yong Choi at the University of Maryland, who gave valuable comments and suggestions and provided continuous support and encouragement in the preparation of this book.

My acknowledgments are due to Sand Duck Lee at McGraw-Hill and Allison Shatkin at CRC Press/Taylor & Francis Group for their strong backing, support, and encouragement with this book effort.

Special thanks are due to Teresita Munoz at CRC Press/Taylor & Francis Group who facilitated the production of this book. Her talents and personal attention are truly appreciated.

I thank Joette Lynch, production editor at Taylor & Francis Group, and Vijay Bose, project manager at SPi Global. They were extremely encouraging, helpful, and supportive throughout. In particular, I appreciate Vijay Bose for his finding mistakes, inconsistencies, and his attention to detail in working with the copyedited proof.

Finally, my special gratitude goes to my wife, Myoung, and my son, Jonathan, who have been my personal inspiration for so many years and have shared the burden of this effort.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

Author

Professor Yeong Koo Yeo teaches chemical engineering at the College of Engineering of Hanyang University, Seoul, South Korea. He received a BA (1979) and an MS (1982) in chemical engineering from Seoul National University and a PhD (1986) in chemical engineering from Auburn University. Before joining the Hanyang faculty in 1993, Professor Yeo had been a senior researcher at KIST. He has taught various courses, such as process control, plant design, process analysis, optimization, computational process design, and model predictive control. Professor Yeo, a leading authority on process control and application of MATLAB, is the author of four books. His book *Introduction and Application of MATLAB* (2009) has played a key role in establishing technical computing as an important field. *Modern Process Control Engineering* (2010) has played a key role in establishing practical control education. Professor Yeo has also authored more than 140 articles in leading academic journals, such as *Industrial and Engineering Chemistry Research*, *IEEE Transactions on Automatic Control*, *Chemical Engineering Communication*, *Japanese Journal of Chemical Engineering*, *Korean Journal of Chemical Engineering*, and *Hwahak Gonhak*. His research interests include process control, process modeling and simulation, process artificial intelligence, and process optimization.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

1 Introduction to MATLAB®

MATLAB® is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numerical computation. MATLAB is one of a number of commercially available, sophisticated mathematical computation tools, which also include Maple, Mathematica, and MathCad. The name MATLAB stands for MATrix LABoratory, because its basic data element is a matrix (Array). MATLAB contains a large number of functions that access proven numerical libraries, such as LINPACK and EISPACK. In fact, MATLAB has many built-in tools for solving problems and developing graphical illustrations. This means that many common tasks can be accomplished with a single function call. MATLAB was originally written in FORTRAN and later rewritten in C, a precursor of C++. The beauty of MATLAB is that we need to know only a tiny bit to get going and be productive. Once we get started, we can pick up new skills quickly with MATLAB's excellent online help features. MATLAB is optimized for matrices. Thus, if a problem can be formulated with a matrix solution, MATLAB executes substantially faster than a similar program in a high-level language.

1.1 STARTING MATLAB®

You can start MATLAB as you would any other software application. In Windows, you access it via the Start menu. Alternatively, you may have a desktop icon that enables you to start MATLAB with a simple double-click. When the MATLAB is started, a window opens in which the main part is the Command Window. [Figure 1.1](#) contains an example of a newly launched MATLAB desktop. In the Command Window, you will see the prompt (>>). If the Command Window is active, the prompt will be followed by a cursor. That is the place where you will enter MATLAB commands. If the Command Window is not active, just click in it anywhere.

In addition to the Command Window, there are a couple of other windows that may be opened by default. The layout can always be customized. To the left of the Command Window is the Current Folder Window. The folder that is set as the Current Folder Window is where files will be saved. This window shows the files that are stored in the Current Folder. To the right of the Command Window are the Workspace Window on top and the Command History Window on the bottom. The Command History Window shows commands that have been entered. The configuration of each window can be altered by clicking the down arrow at the top right corner of the window. This will show a menu of options including closing and undocking that window. Alternatively, hitting the down arrow under Layout in the “HOME” tab allows for the customization of the window within the desktop environment as shown in [Figure 1.2](#).

1.1.1 ENTERING COMMANDS IN THE COMMAND WINDOW

Click in the Command Window to make it active. When a window becomes active, its title bar darkens and a blinking cursor appears after the prompt. Now you can begin entering commands. Try typing `a = [2 4 7]` and press the Enter or Return key:

```
>> a = [2 4 7]
a =
2      4      7
```

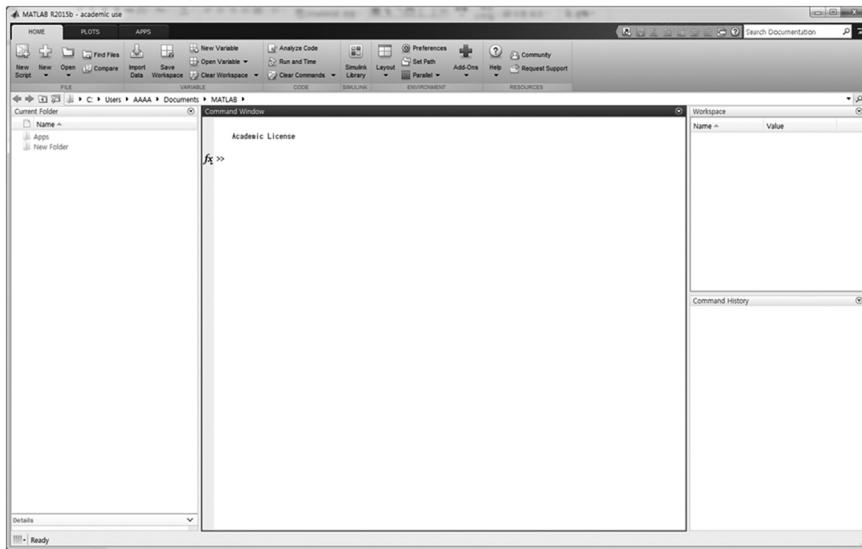


FIGURE 1.1 A MATLAB® desktop.

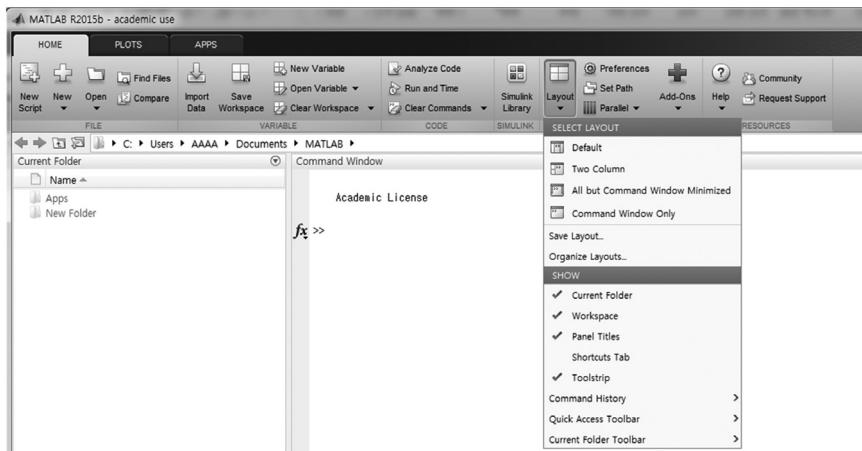


FIGURE 1.2 Customization of the windows using Layout menu.

MATLAB assigns the vector $[2 \ 4 \ 7]$ to a variable of your choice (i.e., *a*). The spacing between the MATLAB commands or expressions and the results can be controlled by entering `format compact`. As another example, type `3+5` and press the Enter key. Then try typing `factor(46738921)` and `sin(80)`. Figure 1.3 shows the results displayed in the Command Window.

Entering a command and manipulating with it require us to master the following rules:

- MATLAB is case sensitive, which means that there is a difference between upper- and lowercase letters.
- A semicolon (`;`) at the end of the command withholds displaying the answer.
- A command in a preceding line cannot be changed; to correct or repeat an executed command, press the up-arrow key (`↑`). Also, an old command can be recalled by typing the first few characters followed by up arrow.

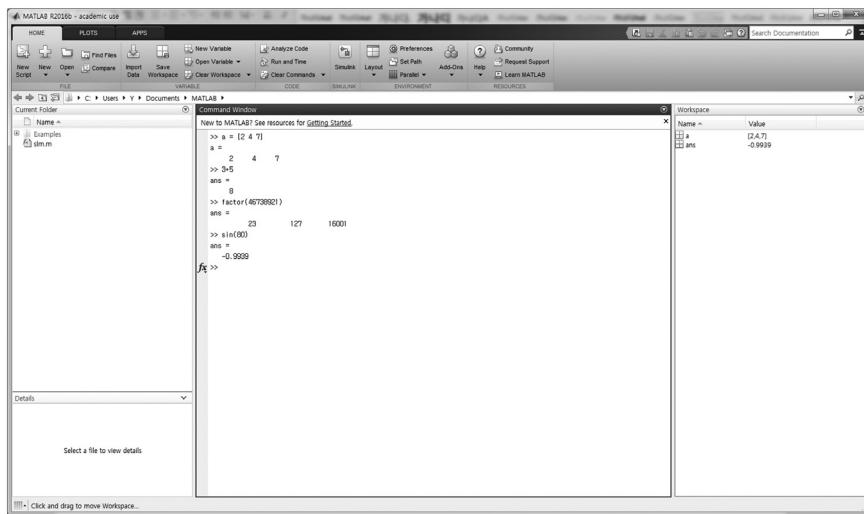


FIGURE 1.3 MATLAB® desktop with several commands evaluated.

- You can type `help topic` to access online help on the command, function, or symbol `topic`.
- If you press the tab key after partially typing a function or variable name, MATLAB will attempt to complete it, offering you a selection of choices if there is more than one possible completion.
- MATLAB uses parentheses (), square brackets [], and curly braces {}, and these are not interchangeable.
- You can quit MATLAB by typing `exit` or `quit` or click the x icon (located at the upper right corner of the MATLAB window.

1.1.2 HELP COMMAND

The `help` command can be used to identify MATLAB functions and also how to use them. To find out what a particular function or command does and how to call it, type `help` and then the name of the function or command. The explanations appear immediately in the Command Window. Alternatively, hitting the down arrow under Help on the toolbar of the Resources group of the desktop “HOME” tab allows for choice of various options as shown in [Figure 1.4](#).

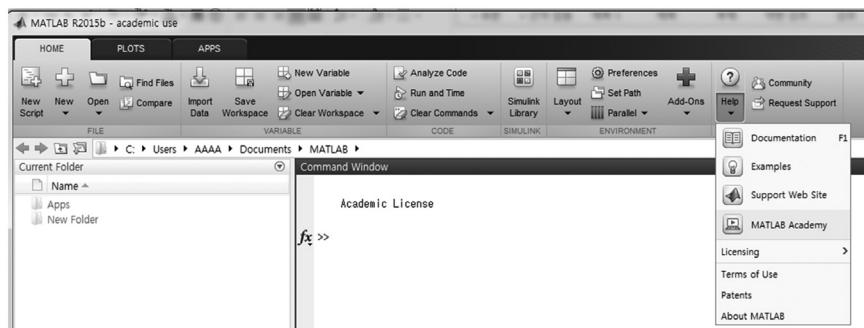


FIGURE 1.4 Submenus of the Help tab.

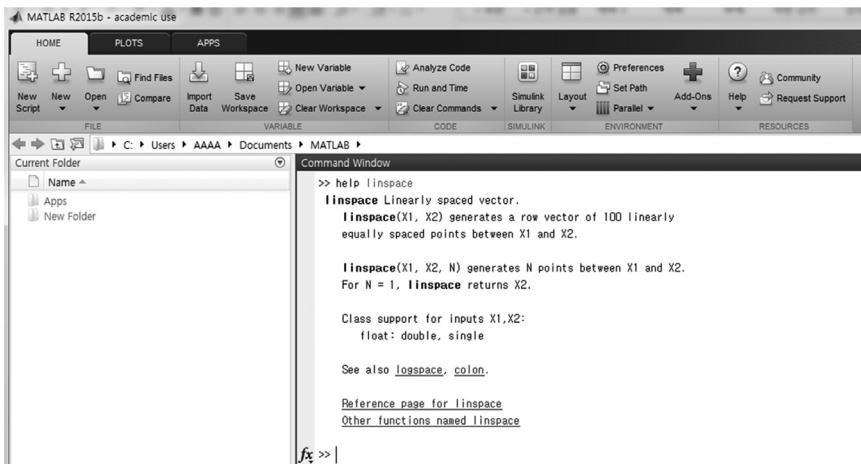


FIGURE 1.5 Description of the linspace command.

For example, the following will give a description of the linspace command as shown in Figure 1.5.

```
>> help linspace
```

1.1.3 EXIT FROM MATLAB®

To exit from MATLAB, either enter `quit` or `exit` at the prompt or click the exit button (located at the upper right corner of the MATLAB window. It is important to save the files or graphics before exiting MATLAB.

1.2 OPERATIONS AND ASSIGNMENT OF VARIABLES

MATLAB can be used as a hand calculator to perform arithmetic operations. MATLAB uses the symbols `+`, `-`, `*`, `/`, and `^` for the addition, subtraction, multiplication, division, and exponentiation (power) of scalars, respectively. For example:

```
>> 6^3 - (7 + 5)/2 + 9*4
ans =
246
>> (-2+sqrt(6))/2
ans =
0.2247
>> 2^(-48)
ans =
3.5527e-15
```

MATLAB assigns the most recent answer to a variable called `ans`, which is an abbreviation for answer. You can use the variable `ans` for further calculations. For example:

```
>> ans^3 + sqrt(ans)
ans =
5.9605e-08
```

TABLE 1.1
Basic Arithmetic Operators Used in MATLAB®

Symbol	Operation	Symbol	Operation
+	Addition	.^	Array exponentiation
-	Subtraction	\	Backslash, left division
*	Multiplication	/	Slash, right division
.*	Array Multiplication	.\	Array left division
^	Exponentiation	./	Array right division

You can use variables to write mathematical expressions. Instead of using the default variable `ans`, you can assign the result to a variable of your own choice. The Workspace Window shows variables that have been created and their values. MATLAB uses the assignment operator (`=`) to create a variable. For example:

```
>> u = cos(18)
u =
    0.6603
>> v = sin(18)
v =
   -0.7510
>> u^2 + v^2
ans =
    1
```

The MATLAB trigonometric functions (`sin(x)`, `cos(x)`, `tan(x)`, etc.) use radian measure. Trigonometric functions ending in d (`sind(x)`, `cosd(x)`, `tand(x)`, etc.) take the argument `x` in degrees. Variable names must begin with a letter; the rest of the name may contain letters, digits, and underscore characters. Variable names can be no longer than 63 characters. MATLAB is case sensitive, which means that there is a difference between upper- and lowercase letters. The variable `pi` is a permanent variable with the value π .

```
>> y = tan(pi/6)
y =
    0.5774
```

Table 1.1 shows the basic arithmetic operators used in MATLAB. Array operations are defined to act elementwise and are generally obtained by preceding the symbol with a dot.

1.2.1 ERRORS IN INPUT

If you make an error in an input line, MATLAB will normally print an error message. For example, here is what happens when you try to evaluate $4x^4$:

```
>> 4x^4
4x^4
↑
Error: Unexpected MATLAB expression.
Did you mean:
>> 4*x^4
```

The error is a missing multiplication operator *. The correct input would be $4*x^4$. Note that MATLAB places a marker (a vertical line segment) at the place where it thinks the error might be. You can edit an input line by using the up-arrow key (\uparrow) to redisplay the previous commands, editing the specific command using the left-arrow key (\leftarrow) and right-arrow key (\rightarrow), and then pressing Return or Enter.

1.2.2 ABORTING CALCULATIONS

If MATLAB gets hung up in a calculation, or seems to be taking too long to perform an operation, you can usually abort it by typing Ctrl+C, that is, holding down the Ctrl key and pressing C. While not foolproof, this is the method of choice when MATLAB is not responding.

1.3 VECTORS AND MATRICES

1.3.1 VECTORS

In MATLAB, a vector is simply a list of scalars, whose order of appearance in the list might be significant. You can create a vector of any length in MATLAB by typing a list of numbers, separated by commas and/or spaces, inside square brackets. For example:

```
>> u = [-2 3 4 8 15]
u =
-2         3         4         8         15
>> u = [-2,3,4,8,15]
u =
-2         3         4         8         15
>> v = [-6 3 -11 9 12 -2 0 5]
v =
-6         3        -11         9         12        -2         0         5
```

If the values in the vector are regularly spaced, the colon operator (:) can be used to iterate through these values. For example:

```
>> t = 0:7
t =
0         1         2         3         4         5         6         7
>> x = -4:3
x =
-4        -3        -2        -1         0         1         2         3
```

Suppose that you want to create a row vector of values running from 1 to 10. Here is how to do it without typing each number:

```
>> w = 1:10
w =
1         2         3         4         5         6         7         8         9         10
```

The increment can be specified as the middle of three arguments. For example, to create a vector with all integers from 8 to 24 in steps of 2:

```
>> y = 8:2:24
y =
8         10        12        14        16        18        20        22        24
```

Increments can be fractional or negative. For example:

```
>> g = 3:3:10, h = 4:-0.75:0
g =
      3       6       9
h =
    4.0000    3.2500    2.5000    1.7500    1.0000    0.2500
```

The elements of the vector w can be extracted as w(1), w(2), etc. For example:

```
>> w = 5:12
w =
      5       6       7       8       9       10      11      12
>> w(4)
ans =
      8
```

A subset of a vector, which would be a vector itself, can also be obtained using the colon operator. For example, the following statement would get the third through sixth elements of the vector w, and store the result in a vector variable v:

```
>> v = w(3:6)
v =
      7       8       9      10
```

Any row vector created using any method can be transposed to result in a column vector. In MATLAB, the apostrophe (') is built in as the transpose operator. For example:

```
>> v'
ans =
      7
      8
      9
     10
```

You can perform mathematical operations on vectors. For example, to square the elements of the vector x, type

```
>> x = 1:2:13
x =
      1       3       5       7       9      11      13
>> x.^2
ans =
      1       9      25      49      81     121     169
```

The dot (period) in this expression says that the numbers in x should be squared individually, or element-by-element. Typing x^2 would tell MATLAB to use matrix multiplication to multiply x by itself and would produce an error message in this case. Similarly, you must type $.*$ or $./$ if you want to multiply or divide vectors element-by-element. For example, to multiply the elements of the vector a by the corresponding elements of the vector b, type

```
>> a = 2:3:14
a =
      2       5       8      11      14
```

```

>> b = [-3 2 -4 1 8]
b =
    -3      2      -4      1      8
>> a.*b
ans =
    -6      10     -32      11     112

```

The `linspace` function creates a linearly spaced vector: `linspace(a,b,n)` creates a vector with `n` variables in the inclusive range from `a` to `b`. If `n` is omitted, the default is `n = 100`. For example:

```

>> linspace(-2,2,8)
ans =
    -2.0000   -1.4286   -0.8571   -0.2857    0.2857    0.8571   1.4286   2.0000

```

MATLAB has many mathematical functions that operate in the array sense when given a vector or matrix argument. These include `exp`, `log`, `sqrt`, `sin`, `cos`, `tan`, etc. For example:

```

>> a = [0.2 1.5 3]
a =
    0.2000      1.5000      3.0000
>> exp(a)
ans =
    1.2214      4.4817     20.0855
>> log(ans)
ans =
    0.2000      1.5000      3.0000
>> sqrt(a)
ans =
    0.4472      1.2247      1.7321

```

If `x` and `y` are column vectors, the dot product or the inner product of these two vectors is accomplished using the `*` operator and transposing the first vector, or by using the `dot` function in MATLAB. The cross product or outer product of two vectors `x` and `y` is defined only when both `x` and `y` have three elements. It can be defined as a matrix multiplication of a matrix composed from the elements of `x` in a particular manner and the column vector `y`. MATLAB has a built-in function `cross` to accomplish this operation. For example:

```

>> x = [-2 0 2]', y = [3 5 7]'
x =
    -2
     0
     2
y =
    3
    5
    7
>> x'*y
ans =
     8
>> dot(x,y)
ans =
     8
>> cross(x,y)
ans =
    -10
    20
    -10

```

The result of the outer product operation x^*y' becomes a matrix:

```
>> Z = x*y'
Z =
-6    -10    -14
 0     0     0
 6    10    14
```

1.3.2 MATRICES

A matrix is a rectangular array of numbers. Row and column vectors are example of matrices. Consider the 3×4 matrix given by

$$A = \begin{bmatrix} -2 & 1 & 3 & 5 \\ 4 & 0 & 6 & -1 \\ 3 & 5 & 2 & 7 \end{bmatrix}$$

It can be entered in MATLAB with the command

```
>> A = [-2 1 3 5; 4 0 6 -1; 3 5 2 7]
A =
-2      1      3      5
 4      0      6     -1
 3      5      2      7
```

The matrix elements in any row are separated by commas or spaces, and the rows are separated by semicolons. If two matrices A and B are the same size, their element-by-element sum is obtained by entering $A+B$. You can also add a scalar c to a matrix A by typing $A+c$. Likewise, $A-B$ represents the difference of A and B, and $A-c$ subtracts the number c from each element of A.

If A and B are multiplicatively compatible, that is, if A is $n \times m$ and B is $m \times 1$, then their product A^*B is $n \times 1$. Recall that the element of A^*B in the i th row and j th column is the sum of the products of the elements from the i th row of A times the elements from the j th column of B. The product of a number c and the matrix A is given by c^*A . A simple illustration is given by the matrix product of the 3×4 matrix A above by the 4×1 column vector x:

```
>> x = [2 -3 4 5]'
x =
 2
 -3
 4
 5
>> A*x
ans =
 30
 27
 34
```

The result is a 3×1 matrix, in other words, a column vector. A' represents the conjugate transpose of A. Consider two 3×3 matrices A and B given by

$$A = \begin{bmatrix} 3 & 1 & 2 \\ -1 & 0 & 1 \\ 6 & 2 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 0 & 1 \\ 1 & -3 & 2 \\ -2 & 1 & 1 \end{bmatrix}$$

Then the transpose of A (A'), addition $C = A+B$, subtraction $D = A-B$, and product $P = A*B$ are given by

```
>> A = [3 1 2; -1 0 1; 6 2 4]
A =
    3      1      2
   -1      0      1
    6      2      4
>> A'
ans =
    3     -1      6
    1      0      2
    2      1      4
>> B = [2 0 1; 1 -3 2; -2 1 1]
B =
    2      0      1
    1     -3      2
   -2      1      1
>> C = A+B
C =
    5      1      3
    0     -3      3
    4      3      5
>> D = A-B
D =
    1      1      1
   -2      3     -1
    8      1      3
>> P = A*B
P =
    3     -1      7
   -4      1      0
    6     -2     14
```

The function `sum` returns the sum of a vector or sums of the columns of a matrix. To find the sum of all elements of a matrix A, we can use `sum(sum(A))`:

```
>> sum(A)
ans =
    8      3      7
>> sum(sum(A))
ans =
    18
```

The function `diag` returns the diagonal in a square matrix:

```
>> diag(A)
ans =
    3
    0
    4
>> sum(diag(A))
ans =
    7
```

For the exponentiation operator, typing A^2 would tell MATLAB to use matrix multiplication to multiply A by itself, $A*A$. But, typing $A.^2$ returns the matrix formed by raising each of the elements of A to the power 2. For example:

```
>> W = [2 1; -3 4]
W =
    2     1
   -3     4
>> W^2
ans =
    1     6
   -18    13
>> W.^2
ans =
    4     1
    9    16
```

In the exponentiation, a vector or a matrix can be used as a power. For example:

```
>> x = [3 1 2], y = [4 2 3], z = [2 1; 4 3]
x =
    3     1     2
y =
    4     2     3
z =
    2     1
    4     3
>> x.^y
ans =
    81     1     8
>> 2.^x
ans =
    8     2     4
>> 2.^z
ans =
    4     2
    16    8
```

As stated before, the conjugate transpose of the matrix A is obtained from A' . If A is real, this is simply the transpose. The transpose without conjugation is obtained with $A.'$. The functional alternatives `ctranspose(A)` and `transpose(A)` are sometimes more convenient. Let Z be a complex matrix given by

$$Z = \begin{bmatrix} -1 & i \\ 2+i & 1+2i \end{bmatrix}$$

```
>> Z = [-1 i; 2+i 1+2i]
Z =
-1.0000 + 0.0000i  0.0000 + 1.0000i
 2.0000 + 1.0000i  1.0000 + 2.0000i
>> Z'
```

```

ans =
-1.0000 + 0.0000i 2.0000 - 1.0000i
0.0000 - 1.0000i 1.0000 - 2.0000i
>> Z.'
ans =
-1.0000 + 0.0000i 2.0000 + 1.0000i
0.0000 + 1.0000i 1.0000 + 2.0000i
>> ctranspose(Z)
ans =
-1.0000 + 0.0000i 2.0000 - 1.0000i
0.0000 - 1.0000i 1.0000 - 2.0000i
>> transpose(Z)
ans =
-1.0000 + 0.0000i 2.0000 + 1.0000i
0.0000 + 1.0000i 1.0000 + 2.0000i

```

The functions `rank`, `det`, and `inv` calculate, respectively, the rank, the determinant, and the inverse of a matrix. The inverse of the matrix B can also be obtained by `B^-1`. For example:

```

>> B = [2 0 1; 1 -2 3; -2 1 1]
B =
2 0 1
1 -2 3
-2 1 1
>> rank(B)
ans =
3
>> det(B)
ans =
-13
>> inv(B)
ans =
0.3846 -0.0769 -0.1538
0.5385 -0.3077 0.3846
0.2308 0.1538 0.3077
>> B^-1
ans =
0.3846 -0.0769 -0.1538
0.5385 -0.3077 0.3846
0.2308 0.1538 0.3077

```

The element in row 2 and column 3 of the matrix B can be accessed as `B(2,3)`. The second column can be extracted by `B(:,2)`, and the third row can be extracted by `B(3,:)`. For example:

```

>> B(2,3)
ans =
3
>> B(:,2)
ans =
0
-2
1
>> B(3,:)
ans =
-2 1 1

```

The submatrix C formed by taking the second and third columns of the matrix B is given by

```
>> C = B(:, 2:3)
C =
    0      1
   -2      3
    1      1
```

MATLAB has several commands that generate special matrices. The command `eye(n)` represents the $n \times n$ identity matrix, and the commands `zeros(n, m)` and `ones(n, m)` generate $n \times m$ matrices of zeros and ones, respectively. The first argument (n) denotes the row and the second argument (m) denotes the column. The commands `eye(3)` and `eye(3, 3)` produce the same identity matrix.

```
>> I3 = eye(3, 3), Y = zeros(3, 5), Z = ones(2)
I3 =
    1      0      0
    0      1      0
    0      0      1
Y =
    0      0      0      0      0
    0      0      0      0      0
    0      0      0      0      0
Z =
    1      1
    1      1
```

The function `rand` creates an array of random numbers between 0 and 1, and the function `randn` creates an array of normal random numbers with mean 0 and variance 1.

```
>> F = rand(3), G = randn(1, 6)
F =
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
G =
    2.7694    -1.3499    3.0349    0.7254    -0.0631    0.7147
```

1.3.3 COMPLEX NUMBER

In MATLAB, `i` and `j` are used as the imaginary unit, $\sqrt{-1}$. The imaginary unit can also be represented by `sqrt(-1)`, or by the function `complex(0, 1)`. For example, the complex number $3-2i$ can be entered as $3-2i$, $3-2*i$, $3-2*sqrt(-1)$, or `complex(3, -2)`. MATLAB supports complex arithmetic, with `conj`, `real`, and `imag` taking the conjugate and the real and imaginary parts from a complex number, respectively. The functions `abs` and `angle` compute the magnitude and the phase angle of a complex number, respectively. For example:

```
>> w = (-1)^0.36
w =
    0.4258 + 0.9048i
>> z = conj(w)
z =
    0.4258 - 0.9048i
>> [real(z) imag(z)]
```

```

ans =
    0.4258    -0.9048
>> exp(i*pi)
ans =
    -1.0000 + 0.0000i
>> angle(w)
ans =
    1.1310
>> abs(w)
ans =
    1

```

The Hermitian transpose of a complex number is obtained by transpose operator ('). For example, the Hermitian transpose of $x = [3 - 2i - 5 + 4i]$ is given by x' . But, $x.'$ produces a simple transpose of x . For example:

```

>> x = [3-2i -5+4i]
x =
    3.0000 - 2.0000i   -5.0000 + 4.0000i
>> x'
ans =
    3.0000 + 2.0000i
   -5.0000 - 4.0000i
>> x.'
ans =
    3.0000 - 2.0000i
   -5.0000 + 4.0000i

```

1.3.4 SUPPRESSION OF SCREEN OUTPUT

Putting a semicolon (;) at the end of an input line suppresses the printing of the output of the MATLAB command. It can be used in any other situation where the MATLAB output need not be displayed. For example, putting a semicolon at the end of the definition of the matrix A suppresses the display of the matrix A on the screen. The result of the operation $A*x$ is not shown in the screen:

```

>> A = [1 2 3 4; 5 6 7 8; 9 10 11 12];
>> A*x;

```

The semicolon should generally be used when defining large vectors or matrices (for example, $x = -1:0.1:2;$).

1.4 NUMERICAL EXPRESSIONS

By default, MATLAB carries out all its arithmetic operations in double-precision floating-point arithmetic, which is accurate to approximately 15 digits. In other words, MATLAB stores floating-point numbers and carries out elementary operations to an accuracy of about 16 significant decimal digits. However, MATLAB displays only four significant decimal digits by default. The `format` command can be used to specify the output format of expressions. To display more digits, type `format long`. Then all subsequent numerical output will have 15 digits displayed. Type `format short` to return to 5-digit display. For example:

```

>> format long
>> v = [1 2 3];
>> sqrt(v)

```

```

ans =
1.000000000000000 1.414213562373095 1.732050807568877
>> format short
>> sqrt(v)
ans =
1.0000 1.4142 1.7321

```

The format command can also be used to control the spacing between the MATLAB command or expression and the result. For example:

```

>> 6.2*4

ans =
24.8000

>> format compact
>> 6.2*4
ans =
24.8000

```

[Table 1.2](#) shows numerical display formats used in MATLAB.

The command `vpa` can be used to do variable-precision arithmetic. For example, to print 50 digits of $\sqrt{5}$, enter:

```

>> vpa('sqrt(5)', 50)
ans =
2.2360679774997896964091736687312762354406183596115

```

If you don't specify the number of digits, the default setting is 32. You can change the default with the command `digits`.

MATLAB has some rounding and remainder functions that are very useful. [Table 1.3](#) shows some of these functions.

For example:

```

>> 53/7
ans =
7.5714
>> round(53/7)
ans =
8

```

TABLE 1.2
Numerical Display Formats

MATLAB Command	Display Format	Example
<code>format</code>	Default: same as <code>format short</code>	
<code>format bank</code>	2 Real decimal digits	3.47
<code>format compact</code>	Suppress redundant line	<code>theta = pi/6</code> <code>theta = 0.5236</code>
<code>format long</code>	14 Decimal digits	3.14159265358979
<code>format short</code>	4 Decimal digits	3.1416
<code>format rat</code>	Fractional form	377/211

TABLE 1.3
Rounding and Remainder Functions

Function	Description
round	Rounds to the nearest integer
ceil	Rounds to the nearest integer toward positive infinity
floor	Rounds to the nearest integer toward minus infinity
fix	Rounds to the nearest integer toward zero

```
>> ceil(53/7)
ans =
    8
>> floor(53/7)
ans =
    7
>> fix(53/7)
ans =
    7
```

1.5 MANAGING VARIABLES

1.5.1 clear COMMAND

The `clc` command clears the command window, leaving a blank page for you to work on. However, this command does not delete from memory the actual variables you have created. The `clear` command deletes all of the saved variables. The action of the `clear` command is reflected in the workspace window. If you want to delete a specific variable, type the variable name right after the `clear` command. For example, if you want to delete the variable `x`, enter

```
>> clear x
```

Table 1.4 shows some options of `clear` command.

1.5.2 COMPUTATIONAL LIMITATIONS AND CONSTANTS

MATLAB includes functions to identify the largest real numbers and the largest integers the program can process. MATLAB also keeps some constants, which are values that are known ahead

TABLE 1.4
Some Options of `clear` Command

Option	Description
<code>clear</code> , <code>clear variables</code>	Deletes all variables from the work space
<code>clear global</code>	Deletes all global variables
<code>clear functions</code>	Deletes all m-files compiled and link to mex files
<code>clear all</code>	Deletes all variables, global variables, functions, mex link, and Java package import list
<code>clear import</code>	Deletes Java package import list (cannot be used within functions)
<code>clear classes</code>	Deletes classes

TABLE 1.5
Typical Computational Limitations and Constants

Limits and Constants	Description	Value
eps	Returns the distance from 1.0 to the next larger floating-point number	2.2204e-16
realmax	Returns the largest possible floating-point number	1.7977e+308
realmin	Returns the smallest possible floating-point number	2.2251e-308
pi	π	3.1415926535897
i, j	Imaginary unit	$\sqrt{-1}$
inf	Infinite number	∞
NaN	Not a number	

of time and cannot possibly change. An example of a constant value would be `pi` (π), which is 3.14159265... [Table 1.5](#) shows some computational limitations and constants.

1.5.3 whos COMMAND

The `whos` command shows variables that have been defined in the Command Window. This command shows more information on the variables compared to the `who` command.

```
>> whos
  Name      Size            Bytes  Class       Attributes
    B         3x3              72  double
    a         1x3              24  double
    b         1x3              24  double
    x         3x3              72  double
```

1.6 SYMBOLIC OPERATIONS

Symbolic operations mean doing mathematical operations on symbols. The symbolic math functions are included in the Symbolic Math Toolbox in MATLAB. Enter `help symbolic` to check whether the Symbolic Toolbox is installed in your system or not. Simple symbolic variables can be created with `sym` and `syms` commands. For example, to create a symbolic variable `y`, enter

```
>> syms y
>> a = 8;
>> a^3 - 4*a*y + y
ans =
512 - 31*y
```

The following creates symbolic variables `x` and `y`, and define `z` as a function of these symbolic variables:

```
>> syms x y
>> z = x^3 - 4*x*y + y^2
z =
x^3 - 4*x*y + y^2
>> 3*y*z
ans =
3*y*(x^3 - 4*x*y + y^2)
```

The `expand` function will multiply out terms and the `factor` function will do the reverse. For example:

```
>> syms x y
>> (x - y) * (x - y)^2
ans =
(x-y)^3
>> expand(ans)
ans =
x^3 - 3*x^2*y + 3*x*y^2 - y^3
>> factor(ans)
ans =
(x-y)^3
```

The `simplify` function simplifies each part of an expression or equation. For example:

```
>> simplify((x^3 - y^3)/(x-y))
ans =
x^2 + x*y + y^2

>> syms p x y
>> y = ((x^p)^(p+1))/x^(p-1);
>> simplify(y)
ans =
x*(x^p)^p
```

Both `syms x` and `x = sym('x')` set the character “x” equal to the symbolic variable x. The `syms` command is particularly convenient, because it can be used to create multiple symbolic variables at the same time. To set the imaginary unit as a symbolic variable, use `sym(sqrt(-1))` or `sym(i)`. For example:

```
>> syms x y sym(i)
>> z = (x-3*i)*(y+4*i)
z =
-(4*i + y)*(3*i - x)
>> expand(z)
ans =
4*i*x - 3*i*y + x*y - 12*i^2
```

1.6.1 SUBSTITUTION IN SYMBOLIC EQUATIONS

Sometimes it is required to substitute numerical values or other symbolic expressions for one or more symbolic variables. The `subs` function will substitute a value for a symbolic variable in an expression. For example, in the following commands, `subs(w, u, 2)` substitutes 2 for the symbolic variable u in the symbolic expression w:

```
>> d = 1, syms u v
d =
1
>> w = u^2 - v^2
w =
u^2 - v^2
>> subs(w,u,2)
ans =
4 - v^2
```

```
>> subs(w,v,d)
ans =
u^2 - 1
>> subs(w,v,u+v)
ans =
u^2 - (u+v)^2
>> simplify(ans)
ans =
-v*(2*u + v)
```

1.7 M-FILES

M-files are ordinary text files containing MATLAB commands. You can create and modify them using any text editor or word processor that is capable of saving files as plain ASCII text. You can use the built-in editor, which you can start by typing `edit`, either by itself (to edit a new file) or followed by the name of an existing M-file in the current directory. The editor window can also be started by hitting the New menu on the toolbar of the FILE group of the desktop “HOME” tab as shown in [Figure 1.6](#). In order to run the M-file created, the Current Folder should be set to the directory where the M-file is stored. There are two different kinds of M-files: script M-files and function M-files.

1.7.1 SCRIPT M-FILES

A script M-file is simply a list of MATLAB statements that are saved in a file with a `.m` file extension. A script M-file contains a sequence of MATLAB commands to be run in order. Open the MATLAB editor and create a file containing the following lines:

```
format long
x = [1.5 0.4 0.06];
y = sin(x) ./x
```

Save this file with the name `scex1.m` in your current directory, or in some directory in your path. You can name the file any way you like, but the “`.m`” suffix is mandatory. You can run this script by

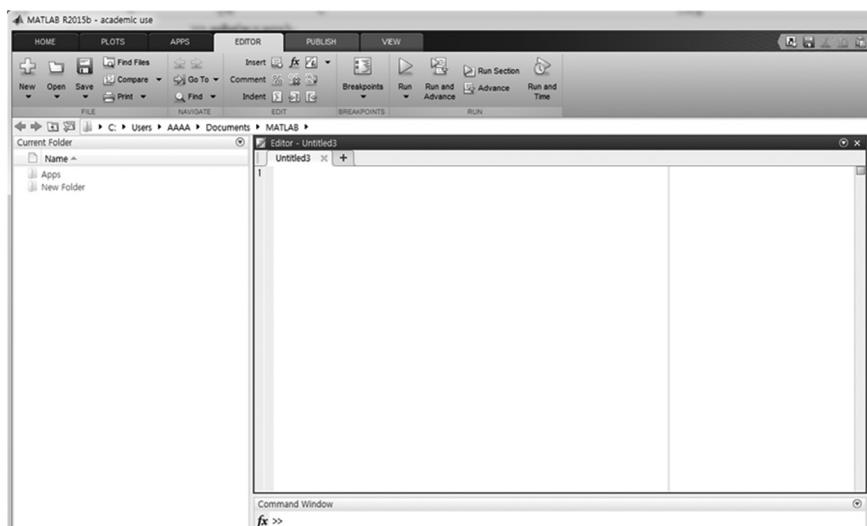


FIGURE 1.6 MATLAB® editor window.

entering `scex1` in the Command Window. The output will be displayed in the Command Window. The M-file can easily be modified. For example, if you wish to calculate $\sin(0.02)/0.02$, you can modify the M-file as follows:

```
format long
x = [1.5 0.4 0.06 0.02];
y = sin(x) ./x
```

Suppose that you want to treat results of examinations with MATLAB functions such as `sort`, `mean`, `median`, and `std`. In the MATLAB editor, type the following commands into an M-file and save this file with the name `mdat.m`:

```
graddat = [15 0 4 26 75 6 48];
exsort = sort(graddat)
exmean = mean(graddat)
exmed = median(graddat)
exstd = std(graddat)
```

You can run this script by entering `mdat` in the Command Window:

```
>> mdat
exsort =
      0      4      6     15     26     48     75
exmean =
    24.8571
exmed =
      15
exstd =
  27.5586
```

1.7.2 ADDING COMMENTS

A comment is an explanatory line for the reader that is ignored by MATLAB when the script is executed. To put in a comment, simply type the `%` character at the beginning of a line or select the comment lines and then click on the `%` symbol on the desktop “EDITOR” tab as shown in [Figure 1.7](#). In the MATLAB editor, the comments will be displayed in green.

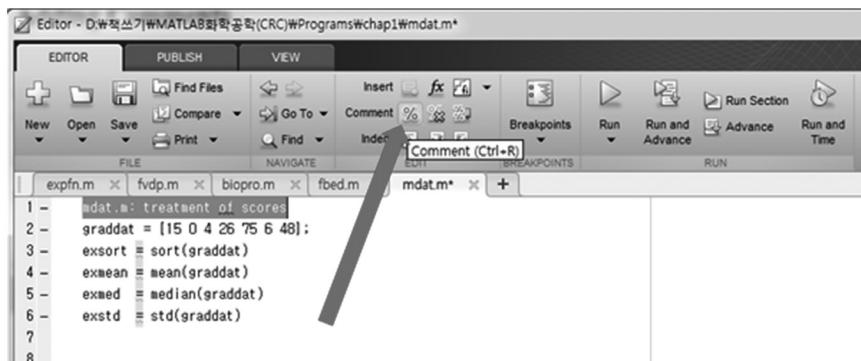


FIGURE 1.7 Adding comments.

The following script is a modified `scex1.m` file with new comments:

```
format long      % 14 decimal digits
x = [1.5 0.4 0.06 0.02];
y = sin(x)./x
% the value of y=sin(x)/x approaches to 1
% as x approaches to zero.
```

Longer comments, called comment blocks, consist of everything in between `%{` and `%}`, which must be alone on separate lines. For example:

```
format long      % 14 decimal digits
x = [1.5 0.4 0.06 0.02];
y = sin(x)./x
%{
    Example of comment block: this is long comments
    the value of y=sin(x)/x approaches to 1
    as x approaches to zero.
%}
```

1.7.3 FUNCTION M-FILES

Function M-files, unlike script M-files, allow you to specify input values when you run them from the MATLAB command line or from another M-file. Like a script M-file, a function M-file is a plain text file that should reside in your current directory or elsewhere in your MATLAB path.

As an example, let's create a function M-file that calculates some values of $\sin(x)/x$ with $x = 10^{-a}$ for several values of a . Here is a function M-file called `sinx` designed to execute these calculations; this file is stored in a file called `sinx.m`.

```
function y = sinx(z)
% SINX calculates sin(x)/x for x = 10^(-a),
% where a = 1, ..., z.
format long
a = 1:z;
x = 10.^(-a);
y = (sin(x)./x)';
end
```

The first line of the file starts with `function`, which identifies the file as a function M-file. The Editor colors this special word blue. The first line of the M-file specifies the name of the function and describes both input arguments (or parameters) and output values. In this example, the function is called `sinx`. The file name without the `.m` extension and the function name should match. It is good practice to follow the first line of a function M-file with one or more comment lines explaining what the M-file does. If you do, `help` will automatically retrieve this information. For example:

```
>> help sinx
sinx calculates sin(x)/x for x = 10^(-a),
where a = 1, ..., z.
```

The following is an example of a call to this function in which z is set to 3 and the value returned is stored in the default variable `ans`:

```
>> sinx(3)
ans =
```

```
0.998334166468282
0.999983333416666
0.999999833333342
```

1.8 FUNCTIONS

MATLAB contains a wide variety of built-in functions. However, you will often find it useful to create your own MATLAB functions.

1.8.1 BUILT-IN FUNCTIONS

MATLAB has many built-in functions such as `sqrt`, `cos`, `sin`, `tan`, `log`, `exp`, `atan`, `gamma`, `erf`, and `besselj`. The function `log` is the natural logarithm that is written as “ln.” The base 10 logarithm is represented by the function `log10` in MATLAB. For example:

```
>> log(exp(5))
ans =
      5
>> log10(exp(5))
ans =
    2.1715
>> sin(2*pi/5)
ans =
    0.9511
```

Symbolic representation can also be used:

```
>> sin(sym('2*pi/5'))
ans =
(2^(1/2)*(5^(1/2) + 5)^(1/2))/4
```

MATLAB also contains many linear algebra functions. Consider a system of linear equations given by

$$-x_1 - 3x_3 = -2$$

$$5x_1 + 2x_2 - 6x_3 = 1$$

$$-4x_1 + x_2 + 8x_3 = 3$$

Now let

$$A = \begin{bmatrix} -1 & 0 & -3 \\ 5 & 2 & -6 \\ -4 & 1 & 8 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad b = \begin{bmatrix} -2 \\ 1 \\ 3 \end{bmatrix}$$

Then the linear system can be expressed as

$$Ax = b$$

The solution to this system can be obtained by using the backslash function (\):

```
>> A = [-1 0 -3; 5 2 -6; -4 1 8]
A =
-1      0      -3
 5      2      -6
-4      1       8
>> b = [-2; 1; 3]
b =
-2
 1
 3
>> x = A\b
x =
 0.4754
 0.8361
 0.5082
```

1.8.2 USER-DEFINED FUNCTIONS

There are three ways to create functions:

1. Use the `inline` command.
2. Create an anonymous function using the `@` operator.
3. Create in a separate M-file.

Anonymous or inline functions are most useful for defining simple functions that can be expressed in one line and for turning the output of a symbolic command into a function. Function M-files are useful for defining functions that require several intermediate commands to compute the output. A separate M-file can be created by using the MATLAB editor as mentioned before. As an example, let's define the function $f(x)=x^3$. We can use the `inline` command:

```
>> f1 = inline('x^3', 'x')
f1 =
  Inline function:
  f1(x) = x^3
```

or we can use the `@` operator to define $f(x)$ as an anonymous function:

```
>> f = @(x) x^3
f =
  function_handle with value:
  @(x)x^3
```

Once the function is created, we can use it by providing the input value:

```
>> f1(5)
ans =
  125
>> f(5)
ans =
  125
```

Vectors and matrices can also be used as input arguments. Thus, it is a good practice to put a dot symbol (.) right before the mathematical operators such as *, /, and ^. The function $f(x)=x^3$ can be redefined using the dot symbol as

```
>> f = @(x) x.^3
f =
  function_handle with value:
  @(x)x.^3
```

or

```
>> f1 = inline('x.^3', 'x')
f1 =
  Inline function:
  f1(x) = x.^3
```

The function defined in this manner can accept a vector as an input argument. For example:

```
>> f(2:8)
ans =
    8     27     64    125    216    343    512
```

We can also define a function of more than two independent variables. For example:

```
>> g = @(x, y) x.^3 + y.^2; g(1,2)
>> g1 = inline('x.^3 + y.^2', 'x', 'y'); g1(1,2)
```

If we want to allow vector–matrix operations in the function, we can define the function $g(x)$ as

```
>> g = @(x, y) x.^3 + y.^2;
```

For example, we can get function values at two points:

```
>> g([1 2], [3 4])
ans =
    10     24
```

1.9 LOOPS

MATLAB has four flow control structures: the `if` statement, the `for` loop, the `while` loop, and the `switch` statement.

1.9.1 if STATEMENT

The simplest form of the `if` statement is

```
if (expression)
  (statements)
end
```

where the `statements` are executed if the elements of `expression` are all nonzero. For example, the following code swaps `x` and `y` if `x` is greater than `y`:

```
if x > y
  temp = y;
```

```

y = x;
x = temp;
end

```

One or more further tests can be added with `elseif`. There must be no space between `else` and `if`.

1.9.2 for Loop

The basic form of the `for` loop is

```

for (variable) = (expression)
    (statements)
end

```

For example, the sum of the first 25 terms of the harmonic series $1/i$ is calculated by

```

>> s = 0;
>> for i = 1:25, s = s + 1/i; end, s
s =
    3.8160

```

Multiple `for` loops can be nested. The `expression` in the `for` loop can be a matrix, in which case `variable` is assigned the columns of `expression` from first to last.

The `while` loop has the form

```

while (expression)
    (statements)
end

```

The `statements` are executed as long as the `expression` is true.

1.10 GRAPHICS

1.10.1 PLOTTING WITH ezplot

The simplest way to graph a function of one variable is with `ezplot`. The `ezplot` function expects a string, a symbolic expression, or an anonymous function, representing the function to be plotted. For example, to plot $x^3 - 4x + 3$ on the interval -3 to 3 using the string form of `ezplot`, enter

```
>> ezplot('x^3 - 4*x + 3', [-3 3])
```

The plot will appear on the screen in a new window labeled “Figure 1.” Using a symbolic expression, you can produce the plot in [Figure 1.8](#) with the following input:

```
>> syms x, ezplot('x^3 - 4*x + 3', [-3 3])
```

An anonymous function can also be used as the argument to `ezplot` as

```
>> ezplot(@(x) x.^3 - 4*x + 3, [-3 3])
```

1.10.2 MODIFYING GRAPHS

A graph can be modified in a number of ways. You can change the title above the graph in [Figure 1.8](#) by typing in the Command Window:

```
>> title 'Plot of a 3rd-order polynomial'
```

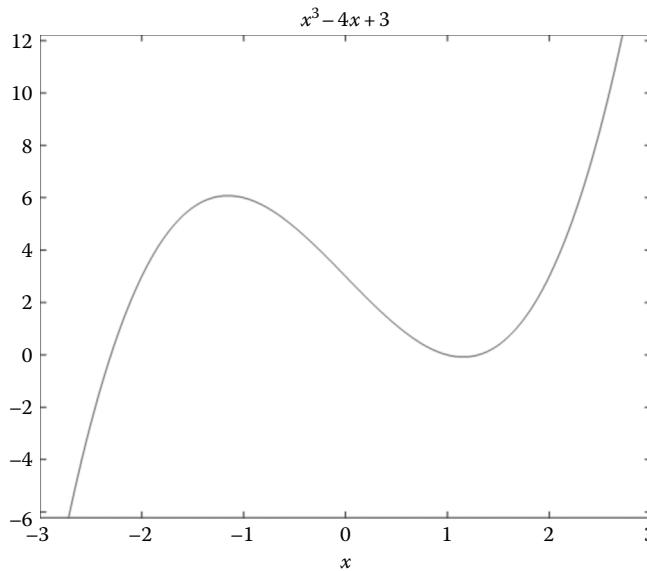


FIGURE 1.8 Plot of $x^3 - 4x + 3$ on the interval $[-3, 3]$.

The same change can be made directly in the figure window by selecting Axes Properties... from the Edit menu at the top of the figure window, as shown in [Figure 1.9](#). You can just type the new title in the box marked “Title.”

You can add a label on the vertical axis with `ylabel` or change the label on the horizontal axis with `xlabel`. Also, you can change the horizontal and vertical ranges of the graph with the `axis` command. For example, to confine horizontal range to the interval from -2 to 2 and the vertical range to the interval from -2 to 8 , type

```
>> axis([-2 2 -2 8])
```

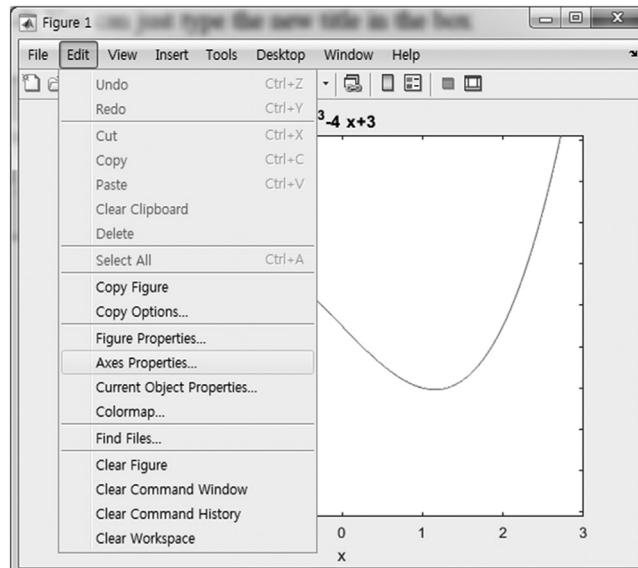


FIGURE 1.9 Edit menu of the figure window.

The first two numbers are the range of the horizontal axis, and the last two numbers are the range of the vertical axis. Both ranges must be included even if only one is changed. To make the shape of the graph square, type `axis square`. This command also makes the scale the same on both axes if the x and y ranges have equal length. For ranges of any lengths, you can force the same scale on both axes without changing the shape by typing `axis equal`.

1.10.3 GRAPHING WITH `plot`

The `plot` function works on vectors of numerical data. The basic syntax is `plot(X, Y)`, where X and Y are vectors of the same length. For example:

```
>> X = [1 2 5]; Y = [4 2 8]; plot(X,Y)
```

The `plot` function considers the vectors X and Y to be lists of the x and y coordinates of successive points on a graph and connects the points with line segments. So, in [Figure 1.10](#), MATLAB connects (1,4) to (2,2) to (5,8).

In general, the `plot` function generates a two-dimensional (2D) graph:

```
>> t = 0:0.002:1; z = exp(9.8*t.* (t-1)).*sin(10*pi*t);
>> plot(t,z)
```

where `plot(t, z)` uses a basic solid line to join the points $t(i)$ with $z(i)$ to create the curve shown in [Figure 1.11](#). The figure window can be closed by typing `close` at the prompt.

To plot $x^3 - 4x + 3$ on the interval $[-2, 2]$, we specify the range of the independent variable x , compute $y = x^3 - 4x + 3$, and then plot the results. The function `grid` introduces a light horizontal and vertical hashing that extends from the axis ticks. The following code produces the plot shown in [Figure 1.12](#).

```
>> x = -2:0.1:2;
>> y = x.^3 -4*x +3;
>> plot(x,y)
>> grid
>> title('x^3 -4*x +3')
>> xlabel('x'), ylabel('y')
```

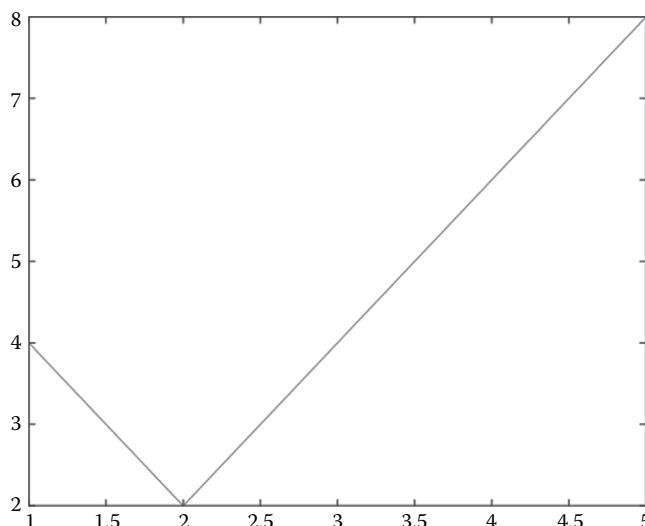


FIGURE 1.10 Plot of line segments.

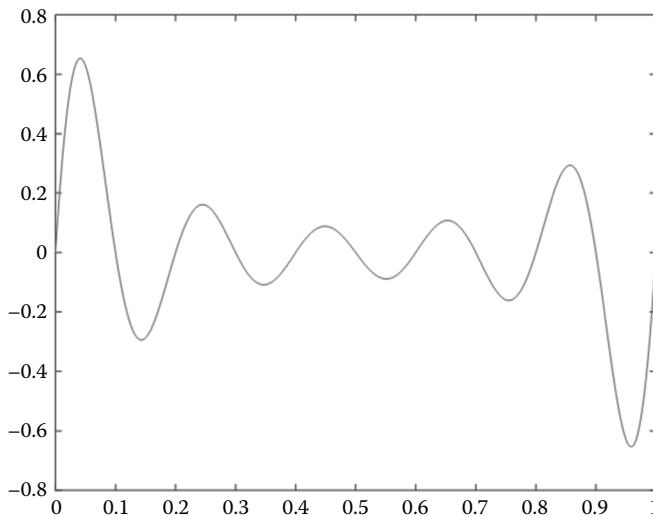


FIGURE 1.11 2D graph by plot function.

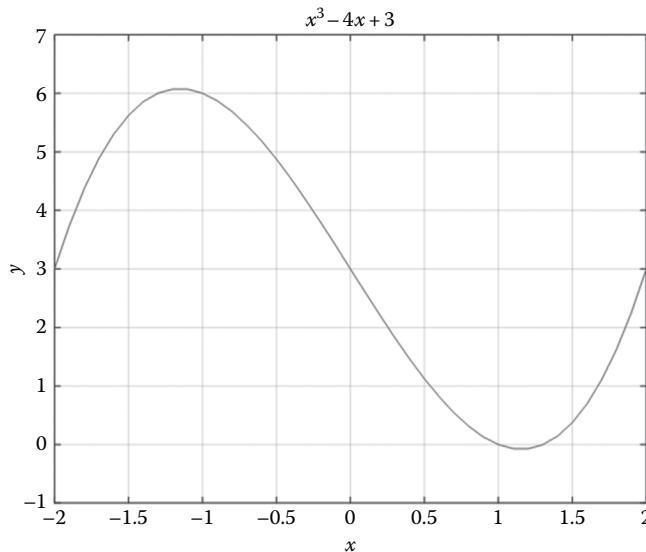


FIGURE 1.12 Graph of $x^3 - 4x + 3$ on the interval $[-2, 2]$.

1.10.4 PLOTTING MULTIPLE CURVES

Each time you execute a plotting command, MATLAB erases the old plot and draws a new one. If you want to overlay two or more plots, use `hold on`. This command instructs MATLAB to retain the old graphics and draw any new graphics on top of the old ones. It remains in effect until you type `hold off`. Do not forget to enter `hold off` after graphing whenever you use `hold on`. The following is an example using the `ezplot` function to create the plot shown in Figure 1.13:

```
>> ezplot('exp(-x^1.2)', [0 6])
>> hold on
```

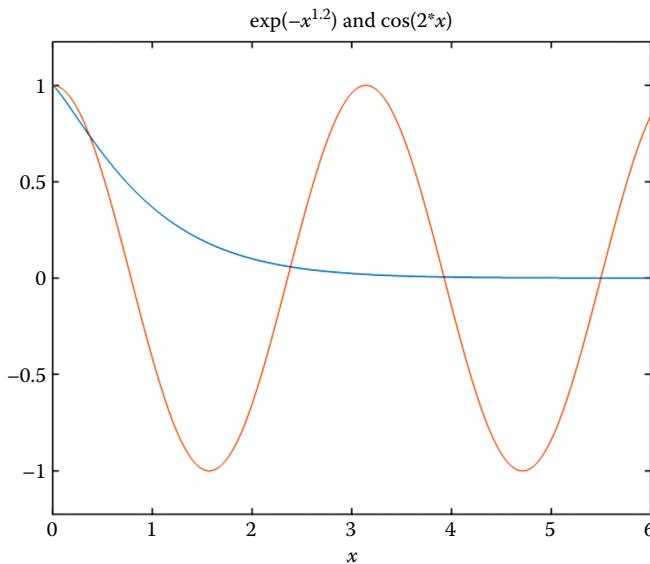


FIGURE 1.13 Plots of $e^{-x^{1.2}}$ and $\cos(2x)$.

```
>> ezplot('cos(2*x)', [0 6])
>> hold off
>> title 'exp(-x^{1.2}) and cos(2*x)'
```

The plot function can also be used to generate the curves shown in [Figure 1.13](#):

```
>> x = 0:0.01:6; plot(x, exp(-x.^1.2), x, cos(2*x))
```

The command `hist` produces a histogram that shows the distribution of data by intervals:

```
>> hist(randn(1000,1))
```

Here, `hist` is given 1000 points from the normal (0,1) random number generator `randn` to create the histogram shown in [Figure 1.14](#).

1.10.5 THREE-DIMENSIONAL PLOTS

For plotting curves in 3-space, the basic command is `plot3`. As an example, plot the following functions on the interval $0 \leq t \leq 8\pi$ using the function `plot3`:

$$x = 0.8 \sin(t), \quad y = 1.2 \cos(t)$$

The following commands generate the three-dimensional (3D) curve shown in [Figure 1.15](#):

```
>> t = [0:0.1:8*pi];
>> x = 0.8*sin(t);
>> y = 1.2*cos(t);
>> plot3(x, y, t)
>> xlabel('x'), ylabel('y'), zlabel('t')
>> grid
```

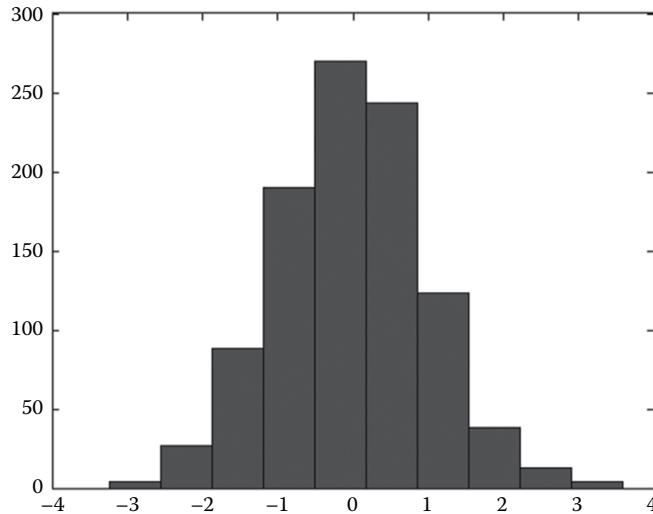


FIGURE 1.14 Histogram by the function hist.

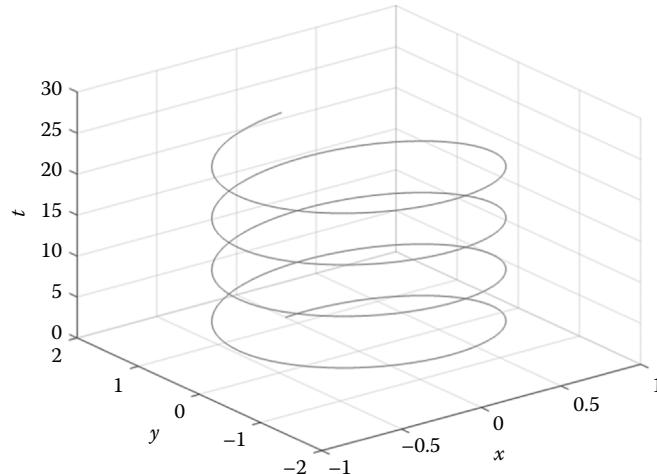


FIGURE 1.15 Example of 3D curve.

1.11 ORDINARY DIFFERENTIAL EQUATIONS

MATLAB has a range of functions for solving ordinary differential equations (ODE). Each of MATLAB's ODE solvers is designed to be efficient in specific circumstances. The function `ode45` implements an adaptive Runge–Kutta algorithm and is typically the most efficient solver. As an example to use the function `ode45`, let's solve the following initial value ODE on the interval $0 \leq t \leq 3$:

$$\frac{dy(t)}{dt} = -y(t) - \sin(3t), \quad y(0) = 1$$

The function file `ode1.m` defines the ODE equation to be solved:

```
function dy = ode1(t,y)
% dy = myode1(t,y) solves ODE.
```

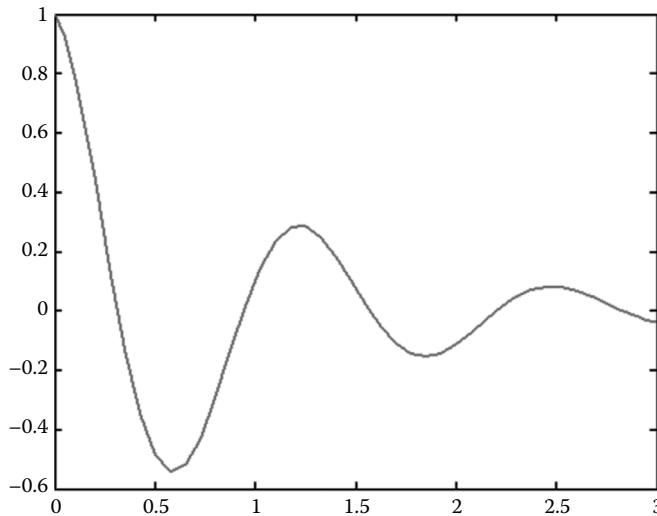


FIGURE 1.16 Solution of simple ODE.

```
dy = - y - 5*exp(-t)*sin(5*t);
end
```

Then, in the Command Window, type:

```
>> tinv = [0 3]; y0 = 1;
>> [t,y] = ode45(@ode1,tinv,y0);
>> plot(t,y)
```

where t_{inv} defines the time interval $0 \leq t \leq 3$ and $y_0=1$ represents the initial condition. The solution is shown in Figure 1.16.

1.12 EXAMPLES

1.12.1 POPULATION GROWTH MODEL

A population growth model can be expressed as $x_{k+1} = ax_k(1 - x_k)$. The following M-file defines this model:

```
function [t, x] = pgmodel(a, xinit, n)
x(1) = xinit; t(1) = 0;
for k = 2:n+1;
    t(k) = k-1;
    x(k) = a*x(k-1)*(1-x(k-1));
end
end
```

This M-file should be stored as `pmod.m`. In the Command Window, specify the values of a ($=2.9$), the initial value ($=0.2$), and the computing time ($=25$), and call the function `pmod`. Figure 1.17 shows the resultant plot.

```
>> [tv, xv] = pgmodel (2.9, 0.2, 25);
>> plot(tv, xv), xlabel('Time'), ylabel('Population')
>> title('Population growth model')
```

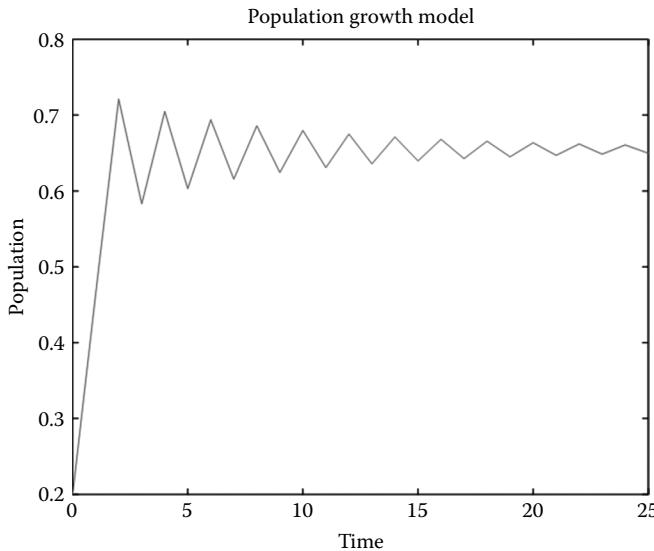


FIGURE 1.17 Population growth curve.

1.12.2 RANDOM FIBONACCI SEQUENCE

A random Fibonacci sequence $\{x_n\}$ is generated by choosing x_1 and x_2 and setting

$$x_{n+1} = x_n \pm x_{n-1}, \quad n \geq 2$$

where \pm indicates that $+$ and $-$ must have equal probability of being chosen. For large n , the quantity $|x_n|$ increases like a multiple of d^n , where $d = 1.13198824\dots$. In the MATLAB editor window, create the script M-file as follows and save it (the file name is `ranfib.m`):

```
% ranfib.m : a random Fibonacci sequence
clear; rand('state',100)      % Set random number state.
m = 1000;                      % number of iterations
x = [1 2];                      % initial condition
for n = 2:m-1                  % for loop
    x(n+1) = x(n) + sign(rand-0.5)*x(n-1);
end
semilogy(1:1000,abs(x))
d = 1.13198824;
hold on
semilogy(1:1000, d.^[1:1000])
hold off
```

Here, the `for` loop stores a random Fibonacci sequence in the array `x`. MATLAB automatically extends `x` each time a new element `x(n+1)` is assigned. The `semilogy` function then plots `n` on the x-axis against `abs(x)` on the y-axis, with logarithmic scaling for the y-axis. Typing `hold on` tells MATLAB to superimpose the next picture on top of the current one. The second `semilogy` function produces a line of slope `d`. Now type

```
>> ranfib
```

at the command line. This will create the graph shown in Figure 1.18.

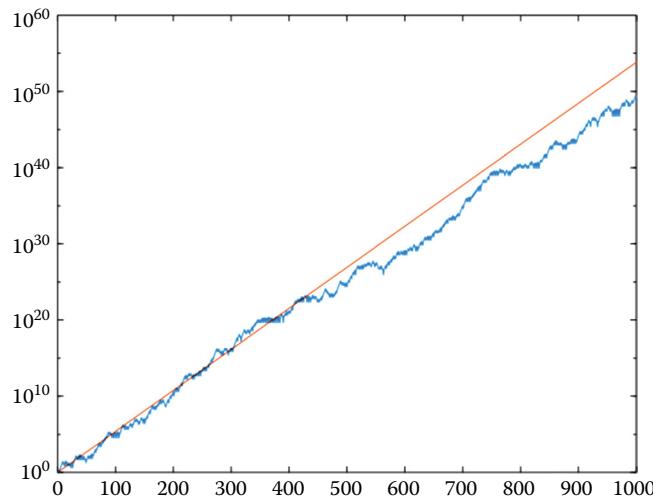


FIGURE 1.18 Random Fibonacci sequence.

1.12.3 GENERATION OF A 3D OBJECT

Let's generate a volume-swept 3D object as shown in [Figure 1.19](#). The script twinobj.m uses the command `surf(X,Y,Z)` to create a 3D surface where the height `Z` is specified at the points `X` and `Y` in the $x-y$ plane.

```
% twinobj.m : generate a volume-swept three-dimensional (3D) object
clear all;
N = 40;          % number of increments
z = linspace(-5,5,N) ';
```

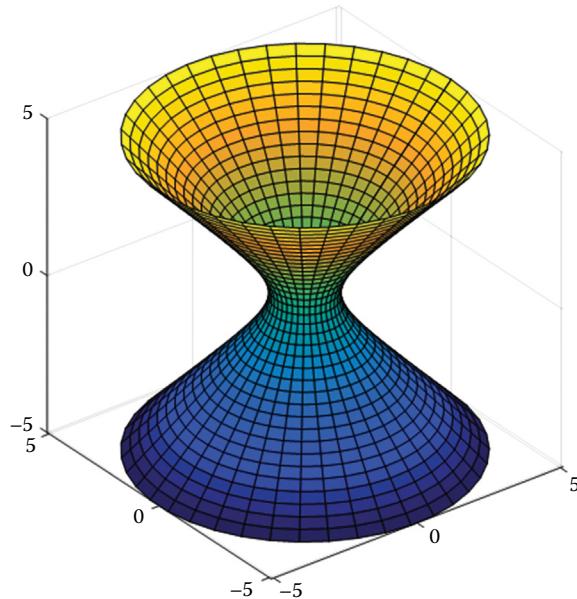


FIGURE 1.19 A volume-swept three-dimensional (3D) object.

```
radius = sqrt(1+z.^2);  
theta = 2*pi*linspace(0,1,N);  
X = radius*cos(theta);  
Y = radius*sin(theta);  
Z = z(:,ones(1,N));  
surf(X,Y,Z)  
axis equal
```

BIBLIOGRAPHY

- Attaway, S., *MATLAB*, 3rd ed., Butterworth-Heinemann, Kidlington, Oxford, UK, 2013.
- Burstein, L., *MATLAB in Quality Assurance Sciences*, Woodhead Publishing, Sawston, Cambridge, UK, 2015.
- Gdeisat, M. and F. Lilley, *MATLAB by Example: Programming Basics*, Elsevier, Waltham, MA, 2013.
- Gilat, A., *MATLAB: An Introduction with Applications*, 4th ed., John Wiley & Sons, Inc., Hoboken, NJ, 2011.
- Hanselman, D. and B. Littlefield, *Mastering MATLAB 7*, Pearson Education International, Upper Saddle River, NJ, 2005.
- Higham, D. J. and N. J. Higham, *MATLAB Guide*, 2nd ed., Siam, Philadelphia, PA, 2005.
- Hunt, B. R., R. L. Lipsman, and J. M. Rosenberg, *A Guide to MATLAB*, 2nd ed., Cambridge Press, 2006.
- Knight, A., *Basics of MATLAB and Beyond*, Chapman & Hall/CRC, Boca Raton, FL, 2000.
- McMahon, D., *MATLAB Demystified*, McGraw-Hill, New York, NY, 2007.
- Moore, H., *MATLAB for Engineers*, 2nd ed., Pearson Educational International, Upper Saddle River, NJ, 2009.
- Palm III, W. J., *Introduction to MATLAB for Engineers*, 3rd ed., McGraw-Hill, New York, NY, 2012.
- Yeo, Y. K., *Introduction to MATLAB Programming* (in Korean), 4th ed., Ajin, Seoul, Korea, 2016.

2 Numerical Methods with MATLAB®

Mathematical models of chemical processes typically involve a complex system of equations. In the field of chemical engineering, most mathematical process models usually consist of linear and non-linear equations, ordinary and partial differential equations, or some combinations of these. In order to investigate the process characteristics, it is necessary to solve the equations using numerical methods. This chapter covers the fundamentals of numerical methods with an emphasis on application using MATLAB®. The topics include solving systems of linear equations, finding roots of equations, polynomial approximation (including interpolation and curve fitting), numerical differentiation and integration, optimization, and solving ordinary and partial differential equations. It is assumed that the reader is already familiar with the underlying theories of the topics covered, so this text will only provide a cursory overview of the theories where necessary.

Numerical methods presented as MATLAB programs are as easy to read and understand as those presented as algorithms or pseudocode. MATLAB also has very convenient graphing capabilities and powerful built-in functions for almost all key numerical methods. Lastly, MATLAB's vector-matrix formulation of operations is an excellent basis for the methods of scientific computing.

This chapter serves as a primer on general programming and algorithm implementation using MATLAB while also presenting MATLAB as a viable problem-solving tool for chemical engineers. MATLAB programs listed in this book are kept as modular as possible to maximize reusability. Much attention has been devoted to the documentation and annotation of the programs developed here.

2.1 LINEAR SYSTEMS

When chemical processes are modeled mathematically, they are occasionally described by systems of linear equations. In this section, we will examine how such equation systems are solved. Let's consider the simultaneous linear equations represented by the following:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

There are n unknown variables x_1, x_2, \dots, x_n and n linear algebraic equations. In general, linear equation systems can be expressed in terms of vectors and matrices as

$$Ax = b$$

where

A is an $n \times n$ matrix of known coefficients

x is the column vector of n unknowns

b is a column vector of n known coefficients as shown below

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

If the coefficient matrix A is square and nonsingular, the solution of the linear system is given by

$$x = A^{-1}b$$

If matrix A is not square (i.e., $A \in \mathbb{R}^{m \times n}$, $m > n$), we cannot compute A^{-1} and the solution cannot be obtained from the above relation. If we multiply A^T on both sides of the linear system, we have

$$A^T A x = A^T b$$

The product $A^T A$ is square, so we can find the inverse of the product. Thus, we obtain

$$x = (A^T A)^{-1} A^T b$$

Letting

$$A^+ = (A^T A)^{-1} A^T$$

we have

$$x = A^+ b$$

The matrix A^+ is called the Moore–Penrose pseudoinverse of A . We can use MATLAB®'s built-in function *pinv* to compute A^+ . In most cases, the solution of linear systems can be found by using the MATLAB backslash operator (\).

Example 2.1: Pseudoinverse of a Matrix

Use MATLAB®'s built-in function *pinv* to compute the pseudoinverse of

$$A = \begin{bmatrix} -2 & -1 & 3 \\ 4 & -5 & 7 \\ 6 & 2 & -5 \\ -3 & 2 & 1 \end{bmatrix}$$

Solution

```
>> A = [-2 -1 3; 4 -5 7; 6 2 -5; -3 2 1];
>> Ap = pinv(A)
Ap =
-0.0109 0.0948 0.1437 0.0878
0.0506 0.0615 0.2103 0.4689
0.0620 0.1294 0.0744 0.2802
```

Example 2.2: Solution of a Linear System Using the Backslash Operator (\)

Use the backslash operator to solve the following equation system:

$$\begin{aligned} 4x_1 + 2x_2 - x_3 &= 8 \\ -3x_1 + x_2 + 2x_3 &= -6 \\ 2x_1 - 4x_2 + x_3 &= 12 \end{aligned}$$

Solution

The solution is given by $x = A \backslash b$. In this problem, the coefficient matrix A is nonsingular and the inverse of A can be used to find the solution.

```
>> A = [4 2 -1; -3 1 2; 2 -4 1];
>> b = [8; -6; 12];
x =
    3
   -1
    2
>> x = inv(A) *b
x =
    3
   -1
    2
```

Example 2.3: Heat Conduction

Figure 2.1 shows a flat square plate with its sides held at constant temperatures. Find the temperature at each node x_1, x_2, x_3 , and x_4 . Each dot represents a node, and the temperature at each node is assumed to be given by the average temperatures of adjacent nodes.

Solution

$$\begin{aligned} x_1 &= \frac{1}{4}(30 + 15 + x_2 + x_3), & x_2 &= \frac{1}{4}(x_1 + 15 + 20 + x_4), & x_3 &= \frac{1}{4}(30 + x_1 + x_4 + 25), \\ x_4 &= \frac{1}{4}(x_3 + x_2 + 20 + 25) \end{aligned}$$

The above relationship can be rearranged in the form of a system of linear equations as

$$4x_1 - x_2 - x_3 = 45, \quad -x_1 + 4x_2 - x_4 = 35, \quad -x_1 + 4x_3 - x_4 = 55, \quad -x_2 - x_3 + 4x_4 = 45$$

or

$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 45 \\ 35 \\ 55 \\ 45 \end{bmatrix}$$

We can use the backslash operator.

```
>> A = [4 -1 -1 0; -1 4 0 -1; -1 0 4 -1; 0 -1 -1 4];
>> b = [45 35 55 45]';
>> x = A\b
x =
    22.5000
    20.0000
    25.0000
    22.5000
```

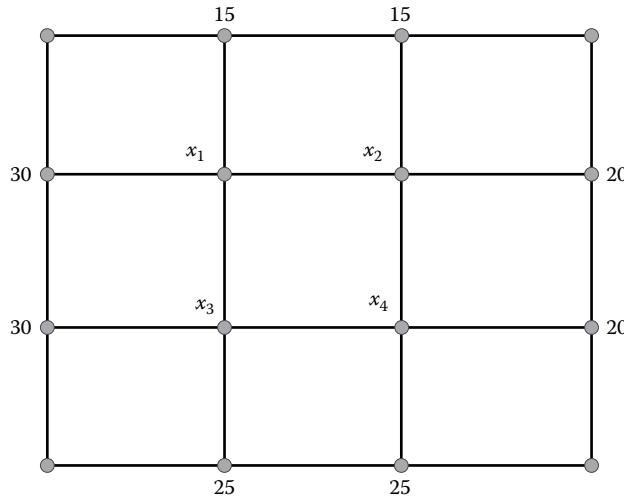


FIGURE 2.1 Flat square plate with heat transfer.

Example 2.4: Two-Dimensional Heat Transfer¹

Consider the cross section of a rectangular flue with steady heat conduction along the x- and y-axes as shown in Figure 2.2. Since both ends of the section are symmetrical, we can consider only 1/8 of the section. From this section, we can construct a node network consisting of small squares.

A heat balance on node 4 gives

$$q_4 = \frac{kA}{\Delta x}(T_5 - T_4) + \frac{kA}{\Delta x}(T_9 - T_4) + \frac{kA}{\Delta y}(T_5 - T_4) + \frac{kA}{\Delta y}(T_9 - T_4)$$

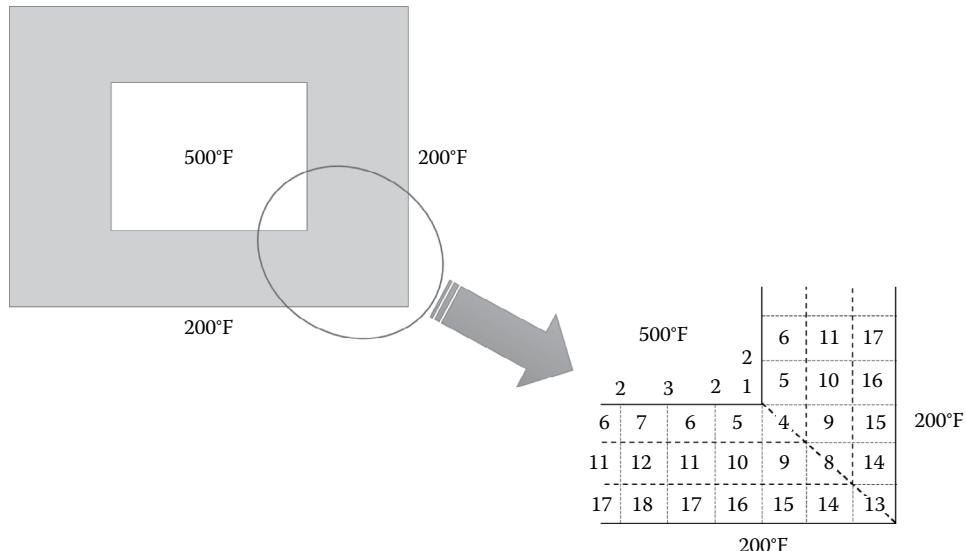


FIGURE 2.2 Cross section of a rectangular flue. (From Kapuno, R.R.A., *Programming for Chemical Engineers*, Infinity Science Press, 2008, p. 122.)

Since each node is square, $\Delta x = \Delta y$, and by rearranging we have

$$\frac{q_4 \Delta x}{kA} = 2T_5 + 2T_9 - 4T_4$$

At steady state, q_4/kA is assumed to be zero. Find temperatures T_4, T_5, \dots, T_{12} at nodes 4 through 12.

Solution

From the heat balance at each node, and considering the boundary conditions, we have

$$T_1 = T_2 = T_3 = 500^{\circ}\text{F}$$

$$T_{13} = T_{14} = T_{15} = T_{16} = T_{17} = T_{18} = 200^{\circ}\text{F}$$

$$\text{At node 4: } -4T_4 + 2T_5 + 2T_9 = 0$$

$$\text{At node 5: } T_4 - 4T_5 + T_6 + T_{10} = -500$$

$$\text{At node 6: } T_5 - 4T_6 + T_7 + T_{11} = -500$$

$$\text{At node 7: } 2T_6 - 4T_7 + T_{12} = -500$$

$$\text{At node 8: } -4T_8 + 2T_9 = -400$$

$$\text{At node 9: } T_4 + T_8 - 4T_9 + T_{10} = -200$$

$$\text{At node 10: } T_5 + T_9 - 4T_{10} + T_{11} = -200$$

$$\text{At node 11: } T_6 + T_{10} - 4T_{11} + T_{12} = -200$$

$$\text{At node 12: } T_7 + 2T_{11} - 4T_{12} = -200$$

The above linear equations can be solved by using the backslash operator as shown below:

```
>> A = [-4 2 0 0 0 2 0 0 0; 1 -4 1 0 0 0 1 0 0; 0 1 -4 1 0 0 0 0 0; ...  
0 0 2 -4 0 0 0 0 1; 0 0 0 -4 2 0 0 0; 1 0 0 0 1 -4 1 0 0; ...  
0 1 0 0 0 1 -4 1 0; 0 0 1 0 0 0 1 -4 1; 0 0 1 0 0 0 0 2 -4];  
>> b = [0 -500 -500 -500 -400 -200 -200 -200 -200]';  
>> x = A\b;  
x =  
311.9011  
369.4600  
387.6418  
391.7217  
227.1712  
254.3423  
278.2969  
289.3855  
291.6032
```

The results are summarized as follows:

$$T_4 = 311.9011^{\circ}\text{F}, \quad T_5 = 369.4600^{\circ}\text{F}, \quad T_6 = 387.6418^{\circ}\text{F}, \quad T_7 = 391.7217^{\circ}\text{F}, \quad T_8 = 227.1712^{\circ}\text{F}
T_9 = 254.3423^{\circ}\text{F}, \quad T_{10} = 278.2969^{\circ}\text{F}, \quad T_{11} = 289.3855^{\circ}\text{F}, \quad T_{12} = 291.6032^{\circ}\text{F}$$

2.2 NONLINEAR EQUATIONS

2.2.1 POLYNOMIAL EQUATIONS

In MATLAB®, a polynomial is represented by a row vector whose elements denote each coefficient sorted in descending powers. The solution of the polynomial equation

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = 0$$

is given by MATLAB's built-in function *roots*. If we define the coefficient vector as $p = [a_n \ a_{n-1} \ \cdots \ a_1 \ a_0]$, the solution is obtained from

```
x = roots(p)
```

Example 2.5: Solution of a Polynomial Equation

Find the roots of the following equation:

$$f(x) = x^5 - 3x^4 + 3x^3 - 2x^2 - 4x + 1 = 0$$

Solution

We can use the function *roots*. But first, we have to construct a coefficient vector with the coefficients sorted in a descending order:

```
>> c = [1 -3 3 -2 -4 1];
>> x = roots(c)
x =
2.3600 + 0.0000i
0.5980 + 1.4065i
0.5980 - 1.4065i
-0.7865 + 0.0000i
0.2306 + 0.0000i
```

Example 2.6: Specific Volume of CO₂

The van der Waals equation of state is given by

$$\left(p + \frac{a}{v^2} \right) (v - b) = RT$$

In this equation, $v = (V/n)$ (n : number of moles), $R = 0.082054$ liter·atm/(mol·K), and $a = 3.592$ and $b = 0.04267$ for CO₂. Find the specific volume (liter/mol) of CO₂ when $P = 12$ atm and $T = 315.6$ K.

Solution

The van der Waals equation can be rearranged as

$$Pv^3 - (bP + RT)v^2 + av - ab = 0$$

which is a 3rd-order polynomial with respect to v . Therefore, the specific volume can be obtained by using the *roots* function:

```
>> P = 12; T = 315.6; R = 0.08205; a = 3.592; b = 0.04267;
>> roots([P - (b*P+R*T) a -a*b])
```

```
ans =
2.0582 + 0.0000i
0.0712 + 0.0337i
0.0712 - 0.0337i
```

Since v should be a real number, we take $v = 2.0582$ liter/mol.

2.2.2 ZEROS OF NONLINEAR EQUATIONS

Several numerical methods can be used to find zeros of nonlinear functions of a single or multiple variables. The following is the overview of several numerical techniques for finding zeros of a nonlinear equation of one variable given by $f(x) = 0$ where $x \in R^1$.

2.2.2.1 Bisection Method

In this method, an initial interval $[a, b]$ is selected first so that $f(a)f(b) < 0$. The midpoint m is calculated by $m = (a + b)/2$. If $f(a)f(m) < 0$, the root is in $[a, m]$ and set $b = m$. Otherwise, the root is in $[m, b]$ and set $a = m$. This process is repeated until the length of the search interval is less than the prescribed small value.

2.2.2.2 False Position Method

The false position method, or the regula falsi method, is similar to the bisection method. Two points $(x_0, f(x_0))$ and $(x_1, f(x_1))$ are selected first so that $f(x_0)f(x_1) < 0$. The intermediate point, x_{k+1} , is calculated by

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k) \quad (k > 0)$$

If $f(x_{k-1})f(x_{k+1}) < 0$, set $x_k = x_{k+1}$; otherwise, set $x_{k-1} = x_{k+1}$. This iteration process is terminated when the root is estimated adequately.

2.2.2.3 Newton's Method

Given an initial estimate of the root, x_0 , and the derivative at x_0 , $f'(x_0)$, the approximation to the root is calculated by

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_0)} \quad (k > 0)$$

This process is repeated until the root is approximated adequately.

2.2.2.4 Secant Method

Two points, $(x_0, f(x_0))$ and $(x_1, f(x_1))$, are selected first. The approximation to the root is given by

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k) \quad (k > 0)$$

This iteration process is terminated when the root is estimated adequately.

2.2.2.5 Muller's Method

For given three points, $(x_0, f(x_0))$, $(x_1, f(x_1))$, and $(x_2, f(x_2))$, we compute

$$m_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}, \quad m_2 = \frac{f(x_2) - f(x_1)}{x_2 - x_1}, \quad n_1 = \frac{m_2 - m_1}{x_2 - x_0}, \quad s = m_2 + n_1(x_2 - x_1)$$

The next approximation is given by

$$x = x_2 - \frac{2f(x_2)}{s + \text{sign}(s)\sqrt{s^2 - 4n_1 f(x_2)}}$$

MATLAB® has a built-in function *fzero* which can be used to find the zeros of a single-variable nonlinear equation $f(x) = 0$. One form of its syntax is

```
x = fzero(@fun, x0)
```

where *fun* is the name of the function $f(x)$ and *x0* is a user-defined guess for the zero. The *fzero* function returns a value of *x* that is near *x0*. This function only identifies the points where the function touches the axis.

Some methods used to find zeros of a single-variable nonlinear equation can be extended to a system of nonlinear equations of multiple variables of a form $F(x) = 0$ where $x \in R^{n \times 1}$ and $F \in R^{n \times 1}$. The following is the overview of some numerical techniques for finding zeros of a nonlinear equation of several variables.

2.2.2.6 Newton's Method for Nonlinear Systems

At each iterative stage, the update for the solution vector, x_{new} , is given by

$$x_{new} = x - J^{-1}F(x)$$

where the Jacobian *J* is

$$J(x_1, \dots, x_n) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

In order to avoid the evaluation of the Jacobian, the following system of linear equations is solved for the vector *z* in practice:

$$J(x)z = -F(x)$$

where $z = x_{new} - x$. The update for Newton's algorithm is then given by

$$x_{new} = z + x$$

2.2.2.7 Fixed-Point Iteration Method for Nonlinear Systems

In this method, the nonlinear system $F(x) = 0$ is converted to the fixed-point form $x = g(x)$. If there is a region *Q* such that $g(Q) \subset Q$ and the Jacobian *G* of *g* satisfies $\|G(z)\|_\infty < 1$, then the iteration $x(k+1) = g(x(k))$ will converge.

The zeros of a system of nonlinear equations can be obtained by using the MATLAB®'s built-in function *fsolve*. Usage of this function is similar to *fzero*.

```
x = fsolve(@fun, x0) or [x, fval] = fsolve(@fun, x0)
```

where *fun* denotes the name of the function defining a system of nonlinear equations and *fval* is the values of the function evaluated at the solution vector *x*.

Example 2.7: Zeros of a Nonlinear Equation

Find the zeros of

$$f(x) = x^3 - x \sin x + 1 = 0$$

Solution

We can use the *fzero* function. To find the zero near -1 :

```
>> fzero(@(x) x^3-x*sin(x)+1, -1)
ans =
-0.7725
```

Example 2.8: Friction Factor Using the Colebrook Equation

The Colebrook equation is given by

$$\frac{1}{\sqrt{f}} = -0.86 \ln \left(\frac{\epsilon/D}{3.7} + \frac{2.51}{N_{Re} \sqrt{f}} \right)$$

Find the friction factor f for $N_{Re} = 6.5 \times 10^4$ and $\epsilon/D = 0.00013$. As the first guess for f , use $f_0 = 0.1$.

Solution

The equation should be rearranged in the form $f(x) = 0$ as

$$\frac{1}{\sqrt{f}} + 0.86 \ln \left(\frac{\epsilon/D}{3.7} + \frac{2.51}{N_{Re} \sqrt{f}} \right) = 0$$

We can find f by using the function *fzero* as follows:

```
>> eD = 1.3e-4; Nre = 6.5e4; f0 = 0.1;
>> Cbfun = @(f) 1/sqrt(f) + 0.86*log(eD/3.7 + 2.51/Nre/sqrt(f));
>> f = fzero(Cbfun,f0)
f =
0.0206
```

Example 2.9: SRK Equation of State

The Soave–Redlich–Kwong (SRK) equation of state is given by

$$P = \frac{RT}{(V-b)} - \frac{a}{V(V+b)\sqrt{T}}$$

where $a = 0.42748 \left(\left(R^2 T_c^{5/2} \right) / P_c \right)$, $b = 0.08664 \left(RT_c / P_c \right)$, and T_c and P_c denote critical temperature (K) and critical pressure (atm) respectively, and V is the specific volume (liter/mol). Find the specific volume of 1-butene at 415 K and 21 MPa. The gas constant $R = 0.082054$ liter·atm/(mol·K) and the critical properties of 1-butene are $T_c = 419.6$ K and $P_c = 40.2$ MPa.

Solution

First, we have to rearrange the SRK equation in the form $f(x) = 0$ as follows:

$$f(V) = \frac{RT}{(V-b)} - \frac{a}{V(V+b)\sqrt{T}} - P = 0$$

Since $1 \text{ MPa} = 9.86923 \text{ atm}$, $P = 207.2538 \text{ atm}$, and $P_c = 396.743 \text{ atm}$. Results are shown below with the first guess of 0.1.

```

>> R = 0.082054; Tc = 419.6; Pc = 396.743; T = 415; P = 207.2538;
>> a = 0.42748*R^2*Tc^2.5/Pc; b = 0.08664*R*Tc/Pc;
>> f = @(V) R*T./(V-b) - a./(V.* (V+b) *sqrt(T)) - P;
>> V0 = 0.1; V = fzero(f,V0)
V =
    0.1291

```

We can see that $V = 0.1291$ liter/mol.

Example 2.10: A System of Nonlinear Equations

Find the zeros of the following three equations:

$$\sin(x) + y^2 + \ln z = 7$$

$$3x + 2y - z^3 = -1$$

$$x + y + z = 5$$

Solution

We start with the definition of the function containing equations in vector form. The initial estimate of a common solution should be set in advance.

```

>> fun = @(x) [sin(x(1))+x(2)^2+log(x(3))-7; 3*x(1)+2*x(2)-x(3)^3+1;
              x(1)+x(2)+x(3)-5];
>> x0 = [0 2 2]';
>> x = fsolve(fun,x0)
Equation solved.
fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.
<stopping criteria details>
x =
    0.6331
    2.3934
    1.9735

```

Example 2.11: Composition of the Equilibrium Mixture²

Ethane reacts with steam to form hydrogen over a cracking catalyst at the temperature of $T = 1000$ K and pressure of $P = 1$ atm. The feed contains 4 moles of steam per mole of ethane. The components present in the equilibrium mixture are shown in Table 2.1. The Gibbs energies of formation (kcal/gmol) of the various components at the reaction temperature (1000 K) are also given in Table 2.1. The equilibrium composition of the effluent mixture is to be calculated using data given in Table 2.1. The initial guess for each component is given in the table.

This problem can be regarded as an optimization problem, which minimizes the total Gibbs energy. The objective function to be minimized is given by

$$\min_{n_i} \frac{G}{RT} = \sum_{i=1}^c n_i \left(\frac{G_i^0}{RT} + \ln \frac{n_i}{\sum n_i} \right)$$

where

n_i is the number of moles of component i

c is the total number of components

R is the gas constant

G_i^0 is the Gibbs energy of pure component i at temperature T

TABLE 2.1
Components Present in Effluent of Ethane-Steam Cracking Reactor

<i>i</i>	Component	Gibbs Energy (kcal/gmol)	Feed (gmol)	Effluent Initial Guess
1	CH ₄	4.61		0.001
2	C ₂ H ₄	28.249		0.001
3	C ₂ H ₂	40.604		0.001
4	CO ₂	-94.61		0.993
5	CO	-47.942		1
6	O ₂	0		0.0001
7	H ₂	0		5.992
8	H ₂ O	-46.03	4	1
9	C ₂ H ₆	26.13	1	0.001

Oxygen, hydrogen, and carbon balances should be set to find n_i .

$$\text{Oxygen: } f_1 = 2n_4 + n_5 + 2n_6 + n_8 - 4 = 0$$

$$\text{Hydrogen: } f_2 = 4n_1 + 4n_2 + 2n_3 + 2n_7 + 2n_8 + 6n_9 - 14 = 0$$

$$\text{Carbon: } f_3 = n_1 + 2n_2 + 2n_3 + n_4 + n_5 + 2n_9 - 2 = 0$$

These balance equations are constraints that can be introduced into the objective function using Lagrange multipliers λ_1 , λ_2 , and λ_3 . The extended objective function is given by

$$\min_{n_i, \lambda_j} F = \sum_{i=1}^c n_i \left(\frac{G_i^0}{RT} + \ln \frac{n_i}{\sum n_i} \right) + \sum_{j=1}^3 \lambda_j f_j$$

All the partial derivatives of the function F with respect to n_i and λ_j vanish at the minimum point. For example, the partial derivative of F with respect to n_i is given by

$$\frac{\partial F}{\partial n_i} = \frac{G_i^0}{RT} + \ln \frac{n_i}{\sum_{k=1}^c n_k} + 1 - \frac{n_i}{\sum_{k=1}^c n_k} + \sum_{j=1}^3 \lambda_j \left(\frac{\partial f_j}{\partial n_i} \right) = 0 \quad (i = 1, 2, \dots, c)$$

$$f_j = 0 \quad (j = 1, 2, 3)$$

Solution

From $f_j = 0$ ($j = 1, 2, 3$), we have

$$f_1 = 2n_4 + n_5 + 2n_6 + n_8 - 4 = 0$$

$$f_2 = 4n_1 + 4n_2 + 2n_3 + 2n_7 + 2n_8 + 6n_9 - 14 = 0$$

$$f_3 = n_1 + 2n_2 + 2n_3 + n_4 + n_5 + 2n_9 - 2 = 0$$

Let $n_s = \sum_{k=1}^c n_k$ in the relation $(\partial F / \partial n_i) = 0$ ($i = 1, 2, \dots, c$) and transform logarithms into exponential functions to avoid taking logarithms of very small numbers. Applying Gibbs energy data given in Table 2.1, we have following nonlinear equations:

$$n_1 - n_s \exp \left[- \left(\frac{4.61 \times 10^3}{RT} + 1 - \frac{n_1}{n_s} + 4\lambda_2 + \lambda_3 \right) \right] = 0,$$

$$n_2 - n_s \exp \left[- \left(\frac{28.249 \times 10^3}{RT} + 1 - \frac{n_2}{n_s} + 4\lambda_2 + 2\lambda_3 \right) \right] = 0,$$

$$n_3 - n_s \exp \left[- \left(\frac{40.604 \times 10^3}{RT} + 1 - \frac{n_3}{n_s} + 2\lambda_2 + 2\lambda_3 \right) \right] = 0,$$

$$n_4 - n_s \exp \left[- \left(\frac{-94.61 \times 10^3}{RT} + 1 - \frac{n_4}{n_s} + 2\lambda_1 + \lambda_3 \right) \right] = 0,$$

$$n_5 - n_s \exp \left[- \left(\frac{-47.942 \times 10^3}{RT} + 1 - \frac{n_5}{n_s} + \lambda_1 + \lambda_3 \right) \right] = 0,$$

$$n_6 - n_s \exp \left[- \left(1 - \frac{n_6}{n_s} + 2\lambda_1 \right) \right] = 0,$$

$$n_7 - n_s \exp \left[- \left(1 - \frac{n_7}{n_s} + 2\lambda_2 \right) \right] = 0,$$

$$n_8 - n_s \exp \left[- \left(\frac{-46.03 \times 10^3}{RT} + 1 - \frac{n_8}{n_s} + \lambda_1 + 2\lambda_2 \right) \right] = 0,$$

$$n_9 - n_s \exp \left[- \left(\frac{26.13 \times 10^3}{RT} + 1 - \frac{n_9}{n_s} + 6\lambda_2 + 2\lambda_3 \right) \right] = 0, \quad R = 1.9872 \text{ (cal/(gmol·K))}$$

There are 12 unknown variables ($n_i (i = 1, 2, \dots, 9)$, $\lambda_j (j = 1, 2, 3)$) and 12 equations. First, we construct the MATLAB® function `rxnfun`, which defines the system of nonlinear equations in vector form. In this function, all the unknown variables are represented in terms of $x_i (i = 1, 2, \dots, 12)$ (i.e., $x_i = n_i (i = 1, 2, \dots, 9)$ and $x_i = \lambda_i (i = 10, 11, 12)$).

```
function f = rxnfun(x,T)
R = 1.9872; ns = sum(x(1:9));
f(1,1) = 2*x(4)+x(5)+2*x(6)+x(8)-4;
f(2,1) = 4*x(1)+4*x(2)+2*x(3)+2*x(7)+2*x(8)+6*x(9)-14;
f(3,1) = x(1)+2*x(2)+2*x(3)+x(4)+x(5)+2*x(9)-2;
f(4,1) = x(1)-ns*exp(-(4.61e3/(R*T)+1-x(1)/ns+4*x(11)+x(12)));
f(5,1) = x(2)-ns*exp(-(28.249e3/(R*T)+1-x(2)/ns+4*x(11)+2*x(12)));
f(6,1) = x(3)-ns*exp(-(40.604e3/(R*T)+1-x(3)/ns+2*x(11)+2*x(12)));
f(7,1) = x(4)-ns*exp(-(-94.61e3/(R*T)+1-x(4)/ns+2*x(10)+x(12)));
f(8,1) = x(5)-ns*exp(-(-47.942e3/(R*T)+1-x(5)/ns+x(10)+x(12)));
f(9,1) = x(6)-ns*exp(-(1-x(6)/ns+2*x(10)));
f(10,1) = x(7)-ns*exp(-(1-x(7)/ns+2*x(11)));
f(11,1) = x(8)-ns*exp(-(-46.03e3/(R*T)+1-x(8)/ns+x(10)+2*x(11)));
f(12,1) = x(9)-ns*exp(-(-26.13e3/(R*T)+1-x(9)/ns+6*x(11)+2*x(12)));
end
```

As the initial estimate, we choose $x_0 = [0.001 \ 0.001 \ 0.001 \ 0.993 \ 1 \ 0.0001 \ 5.992 \ 1 \ 0.001 \ 10 \ 10 \ 10]$. Zeros of the nonlinear equation system can be found by using the built-in function `fsove`. The script file `ethanrxn` shows the solution procedure.

```
% ethanrxn.m: ethane-steam cracking reaction
x0 = [0.001 0.001 0.001 0.993 1 0.0001 5.992 1 0.001 10 10 10]';
T = 1000;
[x, fval] = fsolve(@rxnfun, x0, [], T);
comp = {'CH4','C2H4','C2H2','CO2','CO','O2','H2','H2O','C2H6'};
lamda = {'lambda1','lambda2','lambda3'};
fprintf('\n i\tComp. \t Initial Val.\t\tFinal val.\n');
for k = 1:length(x)
    fprintf('%d\t%s\t%12.9f\t%12.9f\n',k,comp{k},x0(k),x(k));
end
fprintf('\n i\tLambda\tInitial Val.\t\tFinal val.\n');
for i = k+1:length(x)
    fprintf('%g\t%s\t%15.9f\t%15.9f\n',i,lambda{i-length(comp)},x0(i),x(i));
end
```

Running this script gives

```
>> ethanrxn
Equation solved.
fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.
<stopping criteria details>

i      Comp.      Initial Val.      Final val.
1      CH4        0.001000000      0.081717614
2      C2H4       0.001000000      0.000000172
3      C2H2       0.001000000      0.000000000
4      CO2        0.993000000      0.664820888
5      CO         1.000000000      1.253460254
6      O2         0.000100000      0.000000000
7      H2         5.992000000      5.419665106
8      H2O        1.000000000      1.416897971
9      C2H6       0.001000000      0.000000450

i      Lambda      Initial Val.      Final val.
10     lambda1     10.0          24.051830977
11     lambda2     10.0          0.051093350
12     lambda3     10.0          1.168410847
```

2.3 REGRESSION ANALYSIS

2.3.1 INTRODUCTION TO STATISTICS

Let N designate the total number of items in the population under consideration, n the number of items contained in the sample taken from the population ($0 \leq n \leq N$), and m_j ($j = 1, 2, \dots, M$) the number of times the value of x_j occurs. The probability of occurrence is defined by the number of occurrences of x_j divided by the total number of observations as

$$Pr(X = x_j) = \lim_{n \rightarrow N} \frac{m_j}{m} = p(x_j)$$

For a discrete random variable, $p(x_j)$ is called the probability function, and it has the following properties:

$$0 \leq p(x_j) \leq 1, \quad \sum_{j=1}^M p(x_j) = 1$$

The expected value of a discrete random variable is defined as

$$\mu = E[X] = \sum_{j=1}^M x_j p(x_j)$$

The expected value corresponds to the center of gravity of the probability density distribution. For the entire population, the expected value is given by the arithmetic average of the random variable as

$$\mu = E[X] = \frac{1}{N} \sum_{i=1}^N x_i$$

The sample mode of a sample observation is the value that occurs most frequently, and the sample mean (or arithmetic average) is the value obtained by dividing the sum of observations by its total tally:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

The expected value of the sample mean is given by

$$E[\bar{x}] = E\left[\frac{1}{n} \sum_{i=1}^n x_i\right] = \frac{1}{n} \sum_{i=1}^n E[x_i] = \frac{1}{n} \sum_{i=1}^n \mu = \mu$$

The population variance is defined as the expected value of the square of the deviation of random variable X from its expectation:

$$\sigma^2 = E[(X - E[X])^2] = E[(X - \mu)^2]$$

For a discrete random variable, the above definition is equivalent to

$$\sigma^2 = \sum_{j=1}^M (x_j - \mu)^2 p(x_j) = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$$

So the population standard deviation is defined as the positive square root of the population variance:

$$\sigma = \sqrt{\sigma^2}$$

The sample variance s^2 is defined as the arithmetic average of the square of the deviations of x_i from the population mean μ :

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2$$

If μ is not known, \bar{x} is used as an estimate of μ , and the sample variance is given by

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

In fact, the sample variance is an unbiased estimate of the population variance:

$$E[s^2] = \sigma^2$$

The positive square root of the sample variance is called the sample standard deviation:

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

The median is the value in the middle of the data if the number of data points is odd.

In MATLAB®, the sample mean is calculated by the built-in function *mean*, and the sample mode is calculated by the built-in function *mode*. The variance and the standard deviation are calculated by the built-in functions *var* and *std*, respectively. The built-in function *median* returns the median value of the elements of the data.

Example 2.12: Elementary Statistics

Table 2.2 shows the measurements of thermal expansion coefficients of a certain metal. Find the mean, median, mode, variance, and standard deviation of this data.

Solution

```
>> tc = [5.394 5.564 5.654 5.465 5.495 5.404 5.524 5.414...
5.514 5.335 5.614 5.455 5.534 5.524 5.475 5.295...
5.384 5.614 5.554 5.675 5.455 5.554 5.504 5.584];
>> fprintf('Mean = %9.6f', mean(tc))
>> fprintf('\nMedian = %9.6f', median(tc))
>> fprintf('\nMode = %9.6f', mode(tc))
>> fprintf('\nVariance = %9.6f', var(tc))
>> fprintf('\nStandard deviation = %9.6f\n', std(tc))

Mean = 5.499333
Median = 5.509000
Mode = 5.455000
Variance = 0.009405
Standard deviation = 0.096977
```

TABLE 2.2
Thermal Expansion Coefficients

Order	Coefficient	Order	Coefficient	Order	Coefficient
1	5.394	9	5.514	17	5.384
2	5.564	10	5.335	18	5.614
3	5.654	11	5.614	19	5.554
4	5.465	12	5.455	20	5.675
5	5.495	13	5.534	21	5.455
6	5.404	14	5.524	22	5.554
7	5.524	15	5.475	23	5.504
8	5.414	16	5.295	24	5.584

2.3.2 GENERATION OF RANDOM NUMBERS

MATLAB® provides three built-in functions for generating random numbers. The function *rand* generates uniformly distributed random numbers between 0 and 1. The function *randn* generates normally distributed random numbers having a mean of 0 and a standard deviation of 1. The following commands show how to generate an $m \times n$ matrix consisting of random numbers:

```
x = rand(m,n)
x = randn(m,n)
```

To generate a random number x within $[x_a \ x_b]$ (i.e., $x_a < x < x_b$), we use

$$x = x_a + (x_b - x_a) * \text{rand}(m,n)$$

A normally distributed $m \times n$ random matrix having a mean of \bar{x} and a standard deviation of s can be generated by

$$x = \bar{x} + s * \text{randn}(m,n)$$

Random integers can be generated by the built-in function *randi*. Let's construct a 2×3 matrix consisting of random integers from 1 to 30:

```
>> M = randi([1 30], 2, 3)
M =
25 4 19
28 28 3
```

Example 2.13: Generation of Random Numbers

1. Generate a 2×3 matrix "x" of random numbers distributed uniformly between 0 and 1.
2. Generate a 2×3 matrix "y" of normally distributed random numbers.
3. Generate a 3×3 matrix "z" of random numbers distributed uniformly between 3 and 5.

Solution

```
(1)
>> x = rand(2,3)
x =
0.9572 0.8003 0.4218
0.4854 0.1419 0.9157
(2)
>> y = randn(2,3)
y =
1.4090 0.6715 0.7172
1.4172 -1.2075 1.6302
(3)
>> z = 3+(5-3)*rand(3,3)
z =
3.0637 3.1943 3.6342
3.5538 4.6469 4.9004
3.0923 4.3897 3.0689
```

2.3.3 LINEAR REGRESSION ANALYSIS

A linear relationship can be represented as

$$y^* = a_0 + a_1 x$$

where

- y^* is the dependent variable
- x is the independent variable

If we let X be the vector of observations of the independent variable and y be the vector of observations of the dependent variable, then the linear model can be rewritten as

$$y = \alpha + X\beta + u$$

where

- u is the vector of disturbance terms
- α is the y -intercept of the line
- β is the slope of the line

This relation can be extended to include more than one independent variable as follows:

$$y = X_1\beta_1 + X_2\beta_2 + \cdots + X_m\beta_m + u$$

where X_1, X_2, \dots, X_m are the vectors of observations of m independent variables. The vector X_1 can be taken as a vector whose elements are all equal to one. In this case, β_1 becomes the y -intercept. We can rewrite the above relation in a vector-matrix form as

$$y = X\beta + u$$

where $y \in R^{n \times 1}$, $X \in R^{n \times m}$, $\beta \in R^{m \times 1}$, $u \in R^{n \times 1}$ and n denotes the number of observations.

If b denotes a m -element vector, which is an estimate of the parameter vector β , then we can define a vector of residuals as

$$\epsilon = y - Xb = y - \hat{y}$$

Each element of ϵ is a difference between the experimental observation y and the calculated value of \hat{y} using the estimated vector b . Let Φ be a sum of the squared residuals given by

$$\Phi = \epsilon^T \epsilon = (y - Xb)^T (y - Xb)$$

Taking the partial derivative of Φ with respect to b and setting the result to zero, we have

$$\frac{\partial \Phi}{\partial b} = (-X)^T (y - Xb) + (y - Xb)^T (-X) = -2X^T (y - Xb) = 0 \Rightarrow (X^T X)b = X^T y$$

Solving the resulting equation for vector b , we have

$$b = (X^T X)^{-1} X^T y$$

Vector b obtained from the above equation minimizes the objective function Φ . The expected value of b is given by

$$E[b] = E[\beta] + (X^T X)^{-1} X^T E[u] = E[\beta] = \beta$$

We can see that b is the unbiased estimate of β .

2.3.4 POLYNOMIAL REGRESSION ANALYSIS

Suppose that a vector of dependent variables y can be represented as a polynomial of the vector of independent variables x :

$$y = b_1 + b_2x + b_3x^2 + \cdots + b_mx^{m-1}$$

Let's define the matrix of independent variables X as

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{m-1} \end{bmatrix}$$

From the weighted least-squares, we have

$$b_W = (X^T W X)^{-1} X^T W y$$

where W is the weighting matrix. b_w minimizes the objective function J defined by

$$J = (Xb - y)^T W (Xb - y)$$

MATLAB®'s built-in function *polyfit* implements least-squares polynomial regression analysis. The calling syntax is

```
p = polyfit(x, y, n)
```

where x is the vector of independent variables, y is the vector of dependent variables, and n is the order of the polynomial to be fitted. Vector p contains the polynomial's resulting coefficients in a descending order.

The built-in function *lscov* performs weighted least-squares calculations for $Xb = y$. This function can be called as

```
b = lscov(X, y, v)
```

where X is the matrix defined by independent variables, y is the vector of dependent variables, and v denotes the inverse of the weighting matrix W .

Example 2.14: Reaction Temperature Profile

Table 2.3 shows measurements of reaction temperature versus time. Determine the 1st-order, 2nd-order, and 4th-order polynomials to represent this data.

TABLE 2.3
Reaction Temperature versus Time

t (hr)	1	2	3	4	5	6	7	8
T (°C)	50.8	56.4	55.1	60.6	61.5	59.5	54.1	53.8

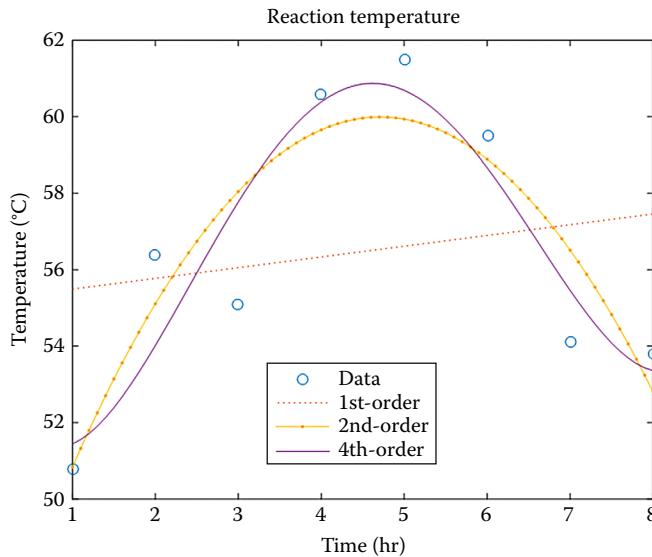


FIGURE 2.3 Polynomial representation of temperature data.

Solution

We can use the built-in function *polyfit* to perform polynomial regressions.

```

>> t = [1 2 3 4 5 6 7 8]; tp = 1:0.1:8;
>> T = [50.8 56.4 55.1 60.6 61.5 59.5 54.1 53.8];
>> p1 = polyfit(t,T,1), p2 = polyfit(t,T,2), p4 = polyfit(t,T,4)
>> T1 = polyval(p1,tp); % Temperature by 1st-order polynomial
>> T2 = polyval(p2,tp); % Temperature by 2nd-order polynomial
>> T4 = polyval(p4,tp); % Temperature by 4th-order polynomial
>> plot(t,T,'o',tp,T1,':',tp,T2,'.-',tp,T4,'-'), title('Reaction
    temperature'),
>> legend('Data','1st-order','2nd-order','4th-order')
>> xlabel('Time(hr)'), ylabel('Temperature(C)')

```

$p1 =$
 $0.2810 \ 55.2107$
 $p2 =$
 $-0.6643 \ 6.2595 \ 45.2464$
 $p4 =$
 $0.0481 \ -0.8684 \ 4.5996 \ -5.8696 \ 53.5357$

The graphs of these lines and the data points are shown in [Figure 2.3](#).

From the results, we can see that the 4th-order polynomial

$$T(t) = 0.0481t^4 - 0.8684t^3 + 4.5996t^2 - 5.8696t + 53.5357$$

is the best-fitting polynomial.

2.3.5 TRANSFORMATION OF NONLINEAR FUNCTIONS TO LINEAR FUNCTIONS

Nonlinear functions can be used in lieu of polynomial functions for fitting functions. Some nonlinear functions can be transformed into linear functions by taking a logarithm or inversion. The resulting linear relation can be used in the linear regression analysis. [Table 2.4](#) summarizes examples of linear transformation of some nonlinear models.

TABLE 2.4
Linearization of Nonlinear Models

Nonlinear Model	Linearization Method	Linear Model
$y = ae^{bx}$	Take natural logarithm on both sides.	$\ln y = \ln a + bx$
$y = ax^b$	Take logarithm on both sides.	$\log y = \log a + b \log x$
$y = \frac{ax}{b+x}$	Invert both sides.	$\frac{1}{y} = \frac{1}{a} + \left(\frac{1}{b}\right)\frac{1}{x}$

Example 2.15: Michaelis–Menten Enzyme Kinetics

An enzyme behaves as a catalyst in a living cell. To represent enzyme-catalyzed reactions, the Michaelis–Menten equation is widely used:

$$r = \frac{a[S]}{b+[S]}$$

where

r is the reaction rate

$[S]$ denotes the concentration of the substrate S

a is the maximum initial reaction rate

b is a constant given by a combination of rate constants

Table 2.5 shows experimental data of reaction rates versus substrate concentrations. Assuming that reaction rates can be represented by the Michaelis–Menten equation, determine parameters a and b of the equation.

Solution

Taking the inverse of the Michaelis–Menten equation $r = ((a[S])/(b + [S]))$, we have

$$\frac{1}{r} = \frac{1}{a} + \frac{b}{a} \frac{1}{S} \Rightarrow y = p_1 x + p_0$$

Coefficients p_1 and p_0 of the linear model can be obtained by using the built-in function *polyfit* with $n = 1$. From the relations $p_0 = 1/a$ and $p_1 = b/a$, we have

$$a = \frac{1}{p_0}, \quad b = p_1 a = \frac{p_1}{p_0}$$

```
>> S = [1.2 1.6 3.2 4.3 5.8 7.6 8.8]; r = [0.06 0.12 0.24 0.27 0.33 0.34 0.34];
>> p = polyfit(1./S, 1./r, 1); % p(1)=p1, p(2)=p0
>> a = 1/p(2), b = p(1)*a
>> Sv = S(1):0.1:S(end);
>> rv = a*Sv./(b + Sv); % Reaction rates by Michaelis-Menten model
>> plot(Sv, rv, S, r, 'o'), title('Michaelis-Menten model'), xlabel('[S]'),
ylabel('r')
>> legend('Michaelis-Menten model', 'Experimental data')
```

TABLE 2.5
Enzyme Reaction Rates versus Substrate Concentrations

[S]	1.2	1.6	3.2	4.3	5.8	7.6	8.8
r	0.06	0.12	0.24	0.27	0.33	0.34	0.34

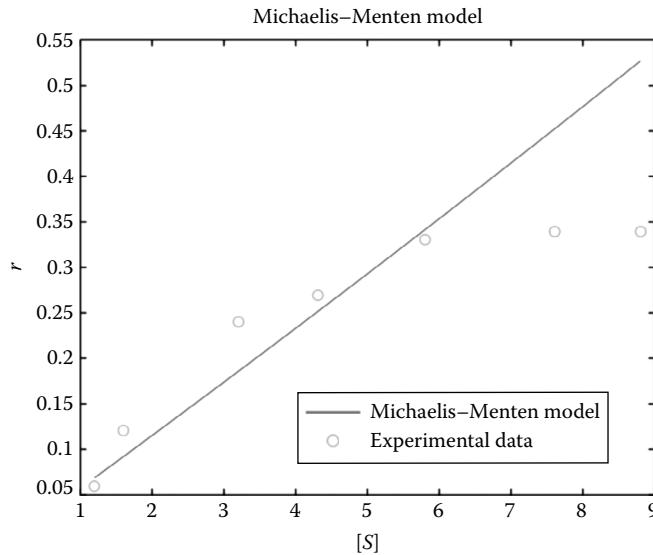


FIGURE 2.4 Michaelis-Menten model.

```
a =
-10.6918
b =
-187.4149
```

We can see that the experimental data can be approximated by

$$r = \frac{-10.6918[S]}{[S] - 187.4149}$$

The graph of this line and the data points are shown in [Figure 2.4](#).

Example 2.16: Vapor Pressure by the Antoine Equation

The Antoine equation is widely used to represent the relationship between vapor pressure and temperature:

$$\log P_v = A + \frac{B}{T + C}$$

P_v is vapor pressure (mmHg)

T is temperature (°C)

A , B , and C are parameters

Use the vapor pressure data given in [Table 2.6](#) to find parameters A , B , and C .

Solution

Multiplying $(T + C)$ on both sides of the Antoine equation, we have

$$(T + C) \log P_v = A(T + C) + B$$

TABLE 2.6
Vapor Pressure versus Temperature

T (°C)	25	27	30	31	35	36	37
P_v (mmHg)	15.8	26.43	39.56	94.91	428.35	861.71	1851.24

By rearranging, we have

$$T \log P_v = AT + (AC + B) - C \log P_v$$

And from dividing both sides by T , we get

$$\log P_v = A + (AC + B) \left(\frac{1}{T} \right) - C \left(\frac{\log P_v}{T} \right) = \begin{bmatrix} 1 & \frac{1}{T} & \frac{\log P_v}{T} \end{bmatrix} \begin{bmatrix} A \\ AC + B \\ -C \end{bmatrix}$$

Let

$$y = \begin{bmatrix} \log P_{v1} \\ \log P_{v2} \\ \vdots \\ \log P_{vn} \end{bmatrix}, \quad X = \begin{bmatrix} 1 & 1/T_1 & \log P_{v1}/T_1 \\ 1 & 1/T_2 & \log P_{v2}/T_2 \\ \vdots & \vdots & \vdots \\ 1 & 1/T_n & \log P_{vn}/T_n \end{bmatrix}, \quad b = \begin{bmatrix} A \\ AC + B \\ -C \end{bmatrix}$$

b is given by

$$b = (X^T X)^{-1} X^T y$$

```

>> T = [25 27 30 31 35 36 37];
>> Pv = [15.8 26.43 39.56 94.91 428.35 861.71 1851.24];
>> X = [ones(1,length(T)); 1./T; log10(Pv)./T]';
>> b = inv(X'*X)*X'*log10(Pv)';
A = b(1), C = -b(3), B = b(2)-A*C
>> Ti = T(1):0.1:T(end); Pvi = 10.^ (A + B./(Ti+C));
>> plot(Ti,Pvi,T,Pv,'o'), xlabel('T(C)'), ylabel('Pv(mmHg)'), legend('Fitting','Data')
A =
-0.0644
C =
-44.2767
B =
-24.9991

```

We can see that the Antoine equation is given by

$$\log P_v = -0.0644 - \frac{24.9991}{T - 44.2767}$$

The graph of this line and the data points are shown in [Figure 2.5](#).

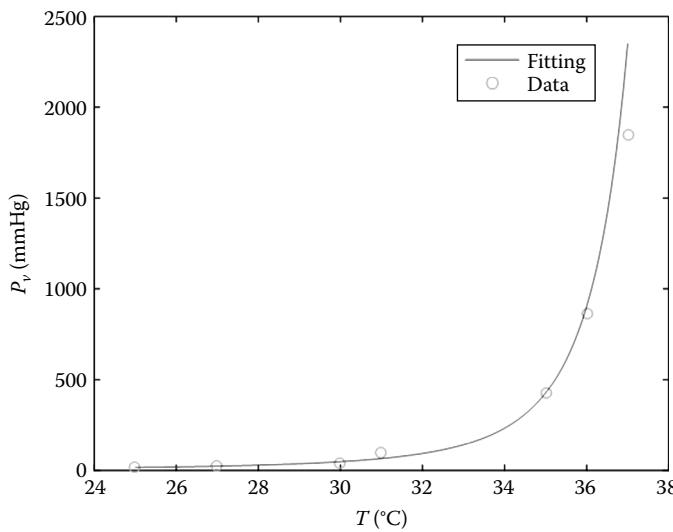


FIGURE 2.5 Antoine vapor pressure model.

Example 2.17: Viscosity of Liquid Propane³

Table 2.7 presents the viscosity (Pa · s) of liquid propane as a function of temperature T (K). A non-linear correlation was proposed to represent the viscosity data:

$$\log(\nu) = A + \frac{B}{T} + C \log T + DT^2$$

Determine parameters A , B , C , and D that best fit this viscosity data.

TABLE 2.7
Viscosity of Liquid Propane

T	$10^4 \nu$ (Pa · s)	T	$10^4 \nu$ (Pa · s)
100	38.2	190	3.36
110	22.9	200	2.87
120	15.3	220	2.24
130	10.8	240	1.80
140	8.32	260	1.44
150	6.59	270	1.30
160	5.46	280	1.17
170	4.53	290	1.05
180	3.82	300	0.959

Source: Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, p. 686, 2008.

Solution

The proposed correlation above can be rearranged as

$$\log(v) = A + \frac{B}{T} + C \log T + DT^2 = \begin{bmatrix} 1 & \frac{1}{T} & \log T & T^2 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}$$

Let

$$y = \begin{bmatrix} \log v_1 \\ \log v_2 \\ \vdots \\ \log v_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & 1/T_1 & \log T_1 & T_1^2 \\ 1 & 1/T_2 & \log T_2 & T_2^2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1/T_n & \log T_n & T_n^2 \end{bmatrix}, \quad b = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}$$

The unknown parameter vector b is given by

$$b = (X^T X)^{-1} X^T y$$

```
>> T = [100 110 120 130 140 150 160 170 180 190 200 220 240 ...  
260 270 280 290 300];  
>> nu = 1e-4*[38.2 22.9 15.3 10.8 8.32 6.59 5.46 4.53 3.82 3.36 ...  
2.87 2.24 1.80 1.44 1.30 1.17 1.05 0.959];  
>> X = [ones(1,length(T)); 1./T; log10(T); T.^2];  
>> b = inv(X'*X)*X'*log10(nu)'; A = b(1), B = b(2), C = b(3), D = b(4);  
>> Tv = T(1):1:T(end); nuv = 10.^(A + B./Tv + C*log10(Tv) + D*Tv.^2);  
>> plot(Tv,nuv,T,nu,'o'), xlabel('T(K)'), ylabel('\nu(Pa*s)'),  
legend('Fitting','Data')  
A =  
-12.6800  
B =  
369.3248  
C =  
3.3279  
D =  
-9.1004e-06
```

The graph of this correlation and the data points are shown in [Figure 2.6](#).

Example 2.18: Catalytic Reaction Model⁴

[Table 2.8](#) presents reaction rate data for heterogeneous catalytic reaction given by $A \rightarrow B$. The following model has been suggested to correlate the data:

$$r = \frac{k_1 P_A}{(1 + K_A P_A + K_B P_B)^2}$$

where k_1 , K_A , and K_B are coefficients to be determined by regression. Calculate the parameters using regression, and plot the reaction rate represented by this model and the rate data.

Solution

Taking the inverse of the given equation, we have

$$\frac{P_A}{r} = \frac{(1 + K_A P_A + K_B P_B)^2}{k_1}$$

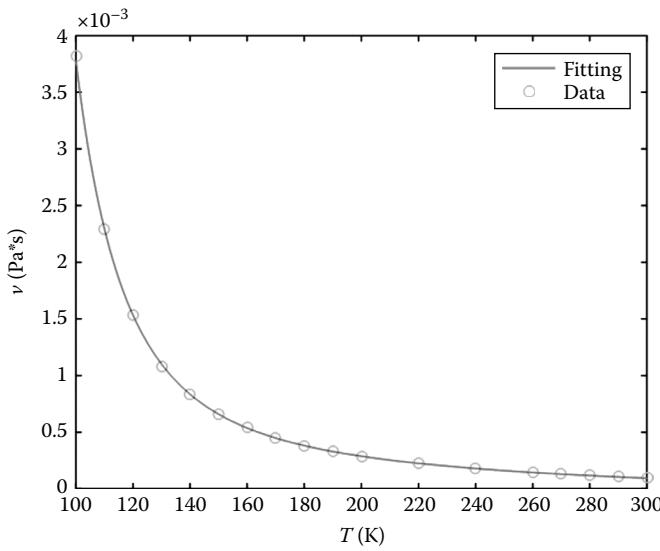


FIGURE 2.6 Correlation of viscosity of liquid propane.

TABLE 2.8
Heterogeneous Catalytic Reaction Rate Data

n	$r \times 10^3$	Partial Pressure (atm)	
		P_A	P_B
1	5.11	1.00	0.00
2	5.42	0.91	0.09
3	5.55	0.82	0.18
4	5.85	0.69	0.31
5	6.02	0.61	0.39
6	6.15	0.52	0.48
7	6.31	0.41	0.59
8	6.45	0.29	0.71

Source: Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, p. 93, 2008.

Rearranging this equation gives:

$$\left(\frac{P_A}{r} \right)^{1/2} = \frac{1}{\sqrt{k_1}} + \frac{K_A}{\sqrt{k_1}} P_A + \frac{K_B}{\sqrt{k_1}} P_B = \begin{bmatrix} 1 & P_A & P_B \end{bmatrix} \begin{bmatrix} 1/\sqrt{k_1} \\ K_A/\sqrt{k_1} \\ K_B/\sqrt{k_1} \end{bmatrix}$$

The unknown parameter vector can be calculated by using the backslash operator (\). Figure 2.7 illustrates the results of comparison between rate data and model.

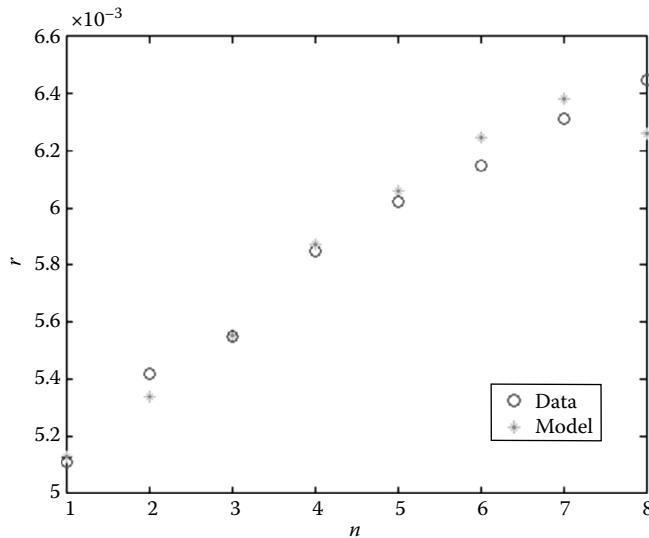


FIGURE 2.7 Comparison between rate data and model.

```

>> r = 1e-3*[5.11 5.42 5.55 5.85 6.02 6.15 6.31 6.45];
>> pA = [1.00 0.91 0.82 0.69 0.61 0.52 0.41 0.29];
>> pB = 1-pA; n = length(r); ni = 1:n;
>> A = [ones(1,n); pA; pB]'; g = sqrt(pA./r)'; b = A\g;
>> k1 = 1/b(1)^2, kA = b(2)/b(1), kB = b(3)/b(1)
>> ri = k1*pA./((1 + kA*pA + kB*pB).^2);
>> plot(ni,r,'o',ni,ri,'*'), legend('data','model'), xlabel('n'), ylabel('r')
Warning: Rank deficient, rank = 2, tol = 5.024296e-15.
k1 =
0.0051
kA =
0
kB =
-0.7219

```

2.4 INTERPOLATION

2.4.1 POLYNOMIAL INTERPOLATION

Sometimes we need to represent a function based on knowledge about its behavior at a set of discrete points. Interpolation is a method that produces a function that best matches a given data, while also providing a good approximation to the unknown values at intermediate points. Suppose that values of a function $f(x_i)$ are known at a set of independent variables x_i as shown in Table 2.9.

TABLE 2.9
Known Function Values

x_i	$f(x_i)$
x_1	$f(x_1)$
x_2	$f(x_2)$
\vdots	\vdots
x_n	$f(x_n)$

The general objective in developing interpolating polynomials is to choose a polynomial $P_n(x)$ of the form

$$P_n(x) = a_0 + a_1x + \cdots + a_nx^n$$

so that this equation best fits the base points of the function, and connect these points with a smooth curve. In short, we find $P_n(x)$ such that

$$P_n(x_i) = f(x_i) \quad (i = 0, 1, \dots, n)$$

The resulting polynomial $P_n(x)$ can be used to estimate data values at intermediate points.

Example 2.19: Determination of Interpolating 2nd-Order Polynomial

Table 2.10 shows an entropy data of saturated steam at three different temperatures. Identify the 2nd-order polynomial that best fits the data, and estimate the entropy at $T = 373.15$ K using this polynomial.

Solution

We have to choose the coefficients of the quadratic equation

$$f(x) = a_2x^2 + a_1x + a_0$$

so that this equation best fits the data points.

The procedure shown in Table 2.11 can be represented as

$$\begin{bmatrix} 313.15^2 & 313.15 & 1 \\ 363.15^2 & 363.15 & 1 \\ 413.15^2 & 413.15 & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 8.2583 \\ 7.4799 \\ 6.9284 \end{bmatrix}$$

TABLE 2.10
Entropy of Saturated Steam

$T(K)$	Entropy (kJ/(kg · K))
313.15	8.2583
363.15	7.4799
413.15	6.9284

TABLE 2.11
Determination of 2nd-Order Polynomial to Fit Entropy Data

i	x_i	$f(x_i) = f_i = a_2x_i^2 + a_1x_i + a_0$
1	313.15	$8.2583 = a_2(313.15)^2 + a_1(313.15) + a_0$
2	363.15	$7.4799 = a_2(363.15)^2 + a_1(363.15) + a_0$
3	413.15	$6.9284 = a_2(413.15)^2 + a_1(413.15) + a_0$

The coefficients can be calculated by the backslash operator.

```
>> format long
>> T = [313.15 363.15 413.15]';
>> A = [T.^2 T ones(3,1)];
>> b = [8.2583 7.4799 6.9284]';
>> x = A\b;
x =
    0.000045380000000
   -0.046258494000000
    18.294051973049942
```

We can see that the entropy of saturated steam can be represented as

$$S(T) = 0.00004538T^2 - 0.0462585T + 18.294052$$

At $T = 373.15$ K, the entropy is calculated as

```
>> s = x(1)*373.15^2 + x(2)*373.15+x(3)
s =
    7.351448
```

From the steam table, we obtain $s = 7.3554$ kJ/(kg·K) at $T = 373.15$ K. Thus, we can see that the entropy of the saturated steam can be adequately estimated by the 2nd-order polynomial.

2.4.2 USE OF THE FUNCTION *polyfit*

The built-in function *polyfit* can be implemented as

```
p = polyfit(x, y, n)
```

where *p* is a vector of coefficients sorted in descending powers, *x* is a vector of independent variables, *y* is a vector of dependent variables, and *n* is the order of the polynomial. When using *polyfit*, the number of data points should be greater than that of polynomial coefficients.

Function values at intermediate points can be calculated using the built-in function *polyval*. For example, the function value at an intermediate point *z* can be found by

```
f = polyval(p, z)
```

where *p* is the vector of coefficients sorted in descending powers, as determined by the function *polyfit*. *f* and *z* may be vectors.

Example 2.20: Polynomial Regression by *polyfit*

Table 2.12 presents the enthalpy of saturated steam versus temperature. Determine a 2nd-order polynomial that fits this enthalpy data, and estimate the enthalpy at $T = 373.15$ K using the polynomial.

Solution

The following commands find coefficients of the 2nd-order polynomial:

```
>> format long
>> T = [283.15 303.15 323.15 363.15 393.15 413.15];
>> H = [2519.9 2556.4 2592.2 2660.1 2706.0 2733.1];
>> p = polyfit(T, H, 2)
>> Tv = 280:0.1:415; Hv = polyval(p, Tv);
```

TABLE 2.12
Enthalpy of Saturated Steam

T (K)	Enthalpy (kJ/kg)
283.15	2519.9
303.15	2556.4
323.15	2592.2
363.15	2660.1
393.15	2706.0
413.15	2733.1

```
>> plot(Tv,Hv,T,H,'o'), legend('2nd-order interpolation','Steam table')
>> xlabel('T(K)'), ylabel('H(kJ/kg)')
p =
1.0e+03 *
-0.00002096326670 0.003108694505382 1.807163153926139
```

We can see that the enthalpy of the saturated steam shown in [Table 2.12](#) can be represented by the quadratic polynomial

$$H(T) = -0.002096327T^2 + 3.108695T + 1807.163154$$

The graph of this curve and the enthalpy data points are shown in [Figure 2.8](#).

The enthalpy of the saturated steam at $T = 350.15$ K can be obtained as

```
>> f = polyval(p, 350.15)
f =
2.638652356432598e+03
```

We have $H = 2638.65$ kJ/kg. From the steam table, we get $H = 2638.7$ kJ/kg at $T = 350.15$ K. Thus, we can see that 2nd-order polynomials can adequately estimate the enthalpy of the saturated steam.

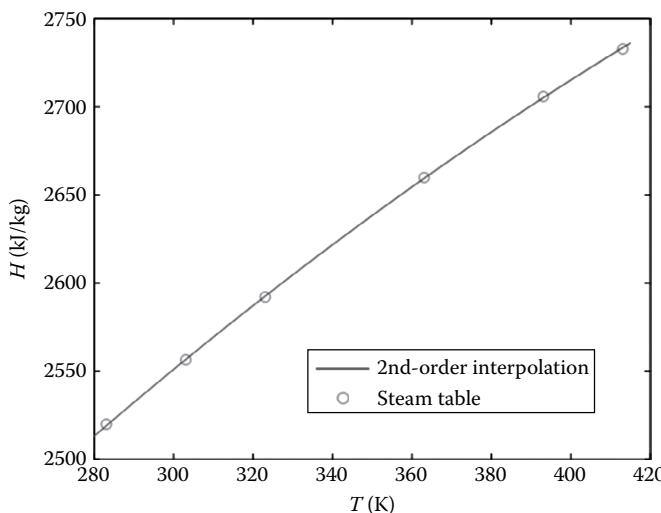


FIGURE 2.8 Interpolation of the enthalpy value.

2.4.3 CUBIC SPLINE INTERPOLATION

We assume that n data points are given: $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. We also assume that $x_1 < x_2 < \dots < x_n$. The cubic polynomial $P_i(x)$ on the interval $x_i \leq x \leq x_{i+1}$ is given by

$$P_i(x) = a_i \frac{(x_{i+1} - x)^3}{h_i} + a_{i+1} \frac{(x - x_i)^3}{h_i} + b_i(x_{i+1} - x) + c_i(x - x_i)$$

where $h_i = x_{i+1} - x_i$. In this relation, b_i and c_i can be represented in terms of a_i as

$$b_i = \frac{y_i}{h_i} - a_i h_i, \quad c_i = \frac{y_{i+1}}{h_i} - a_{i+1} h_i$$

The $n - 2$ unknown parameters $a_i (i = 2, 3, \dots, n - 1)$ can be obtained by solving the following $n - 2$ equations:

$$h_i a_i + 2(h_i + h_{i+1}) a_{i+1} + h_{i+1} a_{i+2} = \frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{y_{i+1} - y_i}{h_i} \quad (i = 1, 2, \dots, n - 2)$$

with $a_1 = a_n = 0$.

MATLAB® provides the built-in function *spline* to perform cubic spline calculations. The basic calling syntax is

```
yi = spline(x, y, xi)
```

where x and y are vectors of independent and dependent variables, respectively, and yi is the vector of the values of the interpolant at xi .

Example 2.21: Cubic Spline Interpolation

Table 2.13 shows experimental data on a pressure drop (kPa) according to flow rates (liter/sec) in a filter. Perform cubic spline interpolation using the built-in function *spline*.

Solution

The following code produces the graph of the interpolation and the data points shown in Figure 2.9.

```
>> x = [0 9.7 15.6 21.3 31.7 35.2 38.4 42.9];
>> y = [0 0.28 0.524 0.998 1.695 2.306 2.781 3.205];
>> xi = min(x):0.1:max(x); yi = spline(x,y,xi);
```

TABLE 2.13
Pressure Drop versus Flow Rates

Flow Rate (liter/sec)	Pressure Drop (kPa)	Flow Rate (liter/sec)	Pressure Drop (kPa)
0	0	31.7	1.695
9.7	0.28	35.2	2.306
15.6	0.524	38.4	2.781
21.3	0.998	42.9	3.205

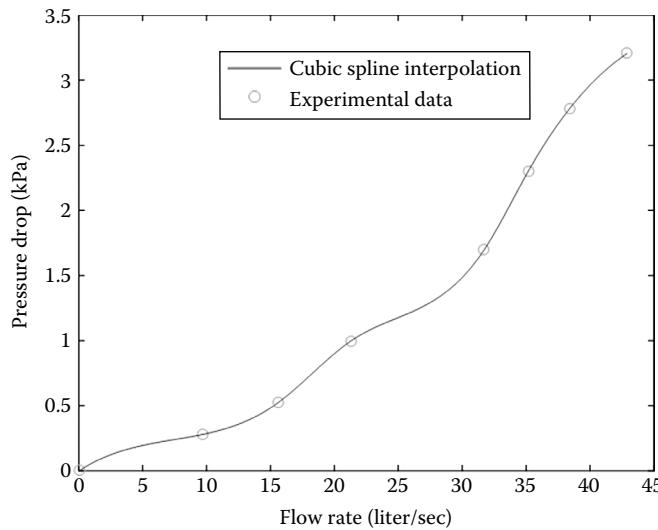


FIGURE 2.9 Cubic spline interpolation.

```
>> plot(xi,yi,x,y,'o'), xlabel('Flow rate(l/s)'), ylabel('Pressure drop(kPa)')
>> legend('Cubic spline interpolation','Experimental data','Location','Best')
```

2.4.4 INTERPOLATION IN ONE DIMENSION

For one-dimensional data, the vector of an independent variable x must be monotonically increasing or decreasing. MATLAB®'s built-in function *interp1* finds the interpolated value y_i for the specified value x_i (x_i and y_i are vectors), based on data given in the vectors x and y . The calling syntax is

```
yi = interp1(x, y, xi, 'method')
```

where 'method' is the desired interpolation method, and the default method is linear interpolation. MATLAB provides several methods as shown in [Table 2.14](#). These methods can also be used in two- and three-dimensional interpolation except *pchip*.

As with cubic splines, the *pchip* method uses cubic polynomials to connect data points with continuous first derivatives. In this method, the second derivatives are not necessarily continuous.

TABLE 2.14
Interpolation Methods Used in the Function *interp1*

Method	Features
Nearest	Nearest neighbor interpolation. This method sets the value of an interpolated point to the value of the nearest existing data point.
Linear	Linear interpolation. This method uses straight lines to connect the points.
Spline	Piecewise cubic spline. This is identical to the <i>spline</i> function.
<i>pchip</i>	Piecewise cubic Hermite interpolation.

Example 2.22: One-Dimensional Interpolation

Determine y at $x = 0.45$ using the benzene–toluene equilibrium data shown in [Table 2.15](#). Try various interpolation methods.

Solution

The commands to implement one-dimensional interpolation are

```
>> x = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1.0];
>> y = [0.00 0.21 0.38 0.51 0.62 0.71 0.79 0.86 0.91 0.96 1.00];
>> lv = interp1(x,y,0.45,'linear');
>> pv = interp1(x,y,0.45,'pchip');
>> sv = interp1(x,y,0.45,'spline');
>> nv = interp1(x,y,0.45,'nearest');
>> fprintf('\nlinear: %6.4f\npchip: %6.4f\nsplne: %6.4f\nnearest: %6.4f\n',
lv,pv,sv,nv);

linear: 0.6650
pchip: 0.6668
splne: 0.6670
nearest: 0.7100
```

Example 2.23: One-Dimensional Fitting

[Table 2.16](#) shows the time series of measurements of reaction temperature. Use MATLAB®'s built-in function *interp1* to fit these data with the pchip (piecewise cubic Hermite) option.

Solution

The commands to implement and plot the pchip interpolation shown in [Figure 2.10](#) are

```
>> tm = [0 18 42 55 70 82 86 95 102 115];
>> TC = [15 23 24 36 78 80 98 96 127 126];
>> tinv = linspace(0,115);
>> yp = interp1(tm, TC, tinv, 'pchip');
>> plot(tm,TC,'o',tinv,yp), xlabel('Time(min)'), ylabel('Temperature(deg.C)')
>> axis([0 115 0 130]), title('cubic Hermite')
```

TABLE 2.15
Benzene–Toluene Equilibrium Data at 1 atm

x	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y	0.00	0.21	0.38	0.51	0.62	0.71	0.79	0.86	0.91	0.96	1.00

TABLE 2.16
Reaction Temperature versus Time

Time (min)	0	18	42	55	70	82	86	95	102	115
Temperature (°C)	15	23	24	36	78	80	98	96	127	126

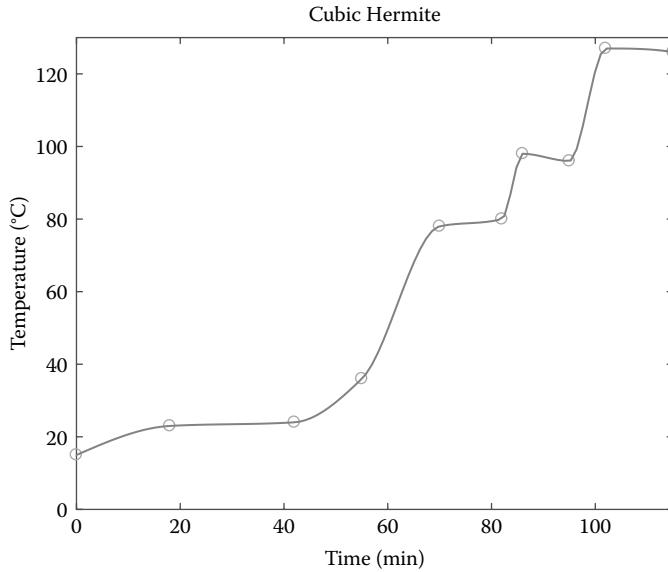


FIGURE 2.10 One-dimensional interpolation (use of *interp1*).

2.4.5 INTERPOLATION IN MULTIDIMENSION

Two-dimensional interpolation deals with determining intermediate values for functions of two independent variables $z = f(x_i, y_i)$. Suppose that we have values at four different points: $z_{11} = f(x_1, y_1)$, $z_{21} = f(x_2, y_1)$, $z_{12} = f(x_1, y_2)$, $z_{22} = f(x_2, y_2)$. Let's interpolate between these points to estimate the function value $z = f(x_i, y_i)$ at an intermediate point (x_i, y_i) ($x_1 \leq x_i \leq x_2, y_1 \leq y_i \leq y_2$). Using the Lagrange interpolation method, the function value at (x_i, y_1) is

$$z_{i1} = f(x_i, y_1) = \frac{x_i - x_2}{x_1 - x_2} z_{11} + \frac{x_i - x_1}{x_2 - x_1} z_{21}$$

and at (x_i, y_2) is

$$z_{i2} = f(x_i, y_2) = \frac{x_i - x_2}{x_1 - x_2} z_{12} + \frac{x_i - x_1}{x_2 - x_1} z_{22}$$

These points can then be used to linearly interpolate along the y-axis to yield

$$\begin{aligned} z = f(x_i, y_i) &= \frac{y_i - y_2}{y_1 - y_2} z_{i1} + \frac{y_i - y_1}{y_2 - y_1} z_{i2} \\ &= \frac{x_i - x_2}{x_1 - x_2} \frac{y_i - y_2}{y_1 - y_2} z_{11} + \frac{x_i - x_1}{x_2 - x_1} \frac{y_i - y_2}{y_1 - y_2} z_{21} + \frac{x_i - x_2}{x_1 - x_2} \frac{y_i - y_1}{y_2 - y_1} z_{12} + \frac{x_i - x_1}{x_2 - x_1} \frac{y_i - y_1}{y_2 - y_1} z_{22} \end{aligned}$$

The MATLAB®'s built-in function *interp2* implements two-dimensional piecewise interpolation. A simple syntax representation of *interp2* is

```
zi = interp2(x, y, z, xi, yi, 'method')
```

where *x* and *y* are matrices containing coordinates of the points at which the values in matrix *z* are given, *zi* is a matrix containing the results of the interpolation as evaluated at the points in the

matrices xi and yi , and 'method' is the desired interpolation method. The methods are identical to those used by *interp1*, that is, nearest, linear, and spline. As with *interp1*, if the 'method' argument is omitted, the default is linear interpolation.

Three-dimensional interpolation deals with determining intermediate values for functions of three independent variables $v = f(x, y, z)$. The MATLAB's built-in function *interp3* implements three-dimensional piecewise interpolation. The calling syntax of *interp3* is

```
Vi = interp3(X, Y, Z, V, Xi, Yi, Zi)
```

where X , Y , and Z are matrices containing the coordinates of the points at which the values in the matrix V are given; Vi is a matrix containing the results of the interpolation as evaluated at the points in the matrices Xi , Yi , and Zi ; and 'method' is the desired interpolation method.

Example 2.24: Two-Dimensional Interpolation

Temperatures are measured at various points on a heated metal plate (Table 2.17). Estimate the temperature at $x_i = 6.4$ and $y_i = 5.2$ using two-dimensional piecewise cubic spline interpolation.

Solution

The commands to implement two-dimensional piecewise cubic spline interpolation are

```
>> x = [2 10]; y = [1 8]; z = [80 78; 75 90];
>> zi = interp2(x, y, z, 6.4, 5.2, 'spline')
zi =
81.5100
```

Example 2.25: Interpolation of Humidity and Dew Point

Table 2.18 presents the absolute humidity (H) and the dew point (DP) of the air as a function of relative humidity (RH). Estimate H and DP when $RH = 58.4$ and T (dry bulb temperature) = 46.8°C using two-dimensional piecewise cubic spline interpolation.

TABLE 2.17
Temperatures at Various Points
on a Heated Metal Plate

<i>x</i>	<i>y</i>	<i>T</i> = $f(x, y)$
2	1	80
2	8	75
10	1	78
10	8	90

TABLE 2.18
Absolute Humidity and Dew Point

Relative Humidity (%RH)	10%	30%	50%	70%	90%	
Dry bulb (°C)						
51	H , g/m ³	8.27	24.75	41.30	58.10	73.51
	DP , °C	10.10	26.89	37.01	43.21	47.80
44	H , g/m ³	6.52	19.58	32.70	45.75	58.78
	DP , °C	6.40	23.34	32.18	38.32	42.81

Solution

The commands to implement two-dimensional spline interpolation are

```
>> RH = [10 30 50 70 90]; T = [51 44];
>> H = [8.27 24.75 41.30 58.10 73.51; 6.52 19.58 32.70 45.75 58.78];
>> DP = [10.10 26.89 37.01 43.21 47.80; 6.40 23.34 32.18 38.32 42.81];
>> Hv = interp2(RH,T,H,58.4,46.8,'spline')
>> DPv = interp2(RH,T,DP,58.4,46.8,'spline')
Hv =
    42.2701
DPv =
    36.9549
```

We can see that the absolute humidity is 42.27 g/m³ and the dew point is 36.95°C when the relative humidity is 58.4% and the dry bulb temperature is 46.8°C.

2.5 OPTIMIZATION

MATLAB® provides two built-in functions for optimization: *fminbnd* and *fminsearch*.

The function *fminbnd* combines golden section search with parabolic interpolation. This function iterates through parabolic interpolations until an acceptable result is obtained. If no results can be obtained, it uses the golden section method. A simple expression of its syntax is

```
[x, fval] = fminbnd(fun, x1, x2)
```

where x1 and x2 are the bounds of the interval being searched, fun is the name of the function being evaluated, x is the location of the minimum, and fval is the function value at x.

The function *fminsearch* can be used to determine the minimum of multidimensional functions. A simple calling syntax of this function is

```
[x, fval] = fminsearch(fun, x0)
```

where x0 is the initial guess (can be a vector or scalar), fun is the name of the function being evaluated, x is the location of the minimum, and fval is the function value at x.

Example 2.26: Use of *fminbnd*

Use *fminbnd* to estimate the minimum of the function $f(x)$ within the interval $0 \leq x \leq 5$:

$$f(x) = -2.5 \sin x + \frac{x^2}{11.8}$$

Solution

The following script estimates the minimum of $f(x)$.

```
>> f = @(x) -2.5*sin(x)+x.^2/11.8;
>> x1 = 0; x2 = 5;
>> [x, fval] = fminbnd(f, x1, x2)
x =
    1.4709
fval =
    -2.3042
```

Example 2.27: Use of *fminsearch*

Use *fminsearch* to estimate the minimum of the function $f(x)$ using the initial guess of $x_0 = [-1, 1]$:

$$f(x_1, x_2) = 3 + 1.2x_1 - x_2 + 2.4x_1^2 + 3x_1x_2 + x_2^2$$

Solution

The following script estimates the minimum of $f(x)$:

```
>> f = @(x) 3+1.2*x(1)-x(2)+2.4*x(1).^2+3*x(1).*x(2)+x(2).^2;
>> x0 = [-1 1];
>> [xopt, fopt] = fminsearch(f, x0)
xopt =
-9.0000 14.0000
fopt =
-9.4000
```

2.6 DIFFERENTIATION AND INTEGRATION

2.6.1 DIFFERENTIATION

The derivative of a function can be estimated by using the function values at a set of discrete points. The difference formulas to estimate the first derivatives are based on the interpolation of a given data set using a straight line. Suppose that a function $f(x)$ and three data points $(x_{i-1}, f(x_{i-1}))$, $(x_i, f(x_i))$, and $(x_{i+1}, f(x_{i+1}))$ are given. The first derivative of $f(x)$ at x_i , $f'(x_i)$, can be estimated by three different formulas.

$$1. \text{ Forward difference formula: } f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

$$2. \text{ Central difference formula: } f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}}$$

$$3. \text{ Backward difference formula: } f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

MATLAB® has two built-in functions to determine the derivatives of data: *diff* and *gradient*. The function *diff* returns a vector containing the differences between adjacent elements. Let x and y be independent and dependent variables, respectively. *diff(y)* computes the differences between the adjacent elements of y , and *diff(x)* computes the differences between the adjacent elements of x . The derivatives of data are given by

```
d = diff(y) ./ diff(x)
```

The *gradient* function evaluates derivatives at the data values themselves, rather than at the intervals between values. A simple expression of its syntax is

```
df = gradient(f, h)
```

where f is a one-dimensional vector and h is the interval (constant) between data points.

Example 2.28: Differentiation by *diff*

Differentiate the function

$$f(x) = 0.3 + 20x - 180x^2 + 650x^3 - 880x^4 + 360x^5$$

from $x = 0$ to 1 using the *diff* function. Compare your results with the exact solution given by

$$f'(x) = 20 - 360x + 1950x^2 - 3520x^3 + 1800x^4$$

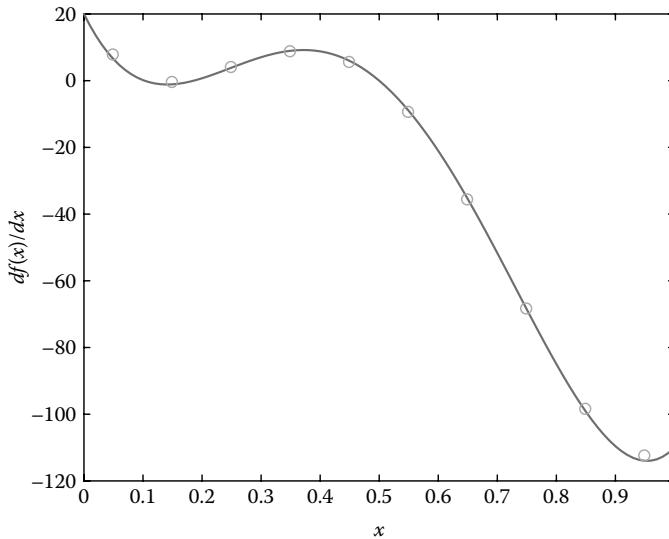


FIGURE 2.11 Differentiation using the built-in function *diff*.

Solution

The following script estimates the differentiation of $f(x)$ and compares the results with the exact solution graphically as shown in Figure 2.11.

```
>> x = 0:0.1:1.0;
>> f = @(x) 0.3+20*x-180*x.^2+650*x.^3-880*x.^4+360*x.^5;x = 0:0.1:1.0;
>> x = 0:0.1:1.0; n = length(x);
>> y = f(x); dr = diff(y)./diff(x)
dr =
7.6560 -0.5840 3.8960 8.6160 5.4160 -9.5440 -35.7840
-68.5040 -98.5840 -112.5840
>> xm = (x(1:n-1) + x(2:n))/2; % the midpoint of each interval
>> x1 = 0:0.01:1; y1 = 20-360*x1+1950*x1.^2-3520*x1.^3+1800*x1.^4;
>> plot(x1, y1, xm, dr, 'o'), xlabel('x'), ylabel('df(x)/dx')
```

Example 2.29: Differentiation by gradient

Differentiate the function

$$f(x) = 0.3 + 20x - 180x^2 + 650x^3 - 880x^4 + 360x^5$$

from $x = 0$ to 1 using the *gradient* function. Compare your results with the exact solution given by

$$f'(x) = 20 - 360x + 1950x^2 - 3520x^3 + 1800x^4$$

Solution

The following script estimates the differentiation of $f(x)$ and computes the exact solution.

```
>> f = @(x) 0.3+20*x-180*x.^2+650*x.^3-880*x.^4+360*x.^5;
>> df = @(x) 20-360*x+1950*x.^2-3520*x.^3+1800*x.^4;
>> x = 0:0.1:1.0; y = f(x);
>> dr = gradient(y, 0.1), dy = df(x) % dr: estimation by gradient, dy: exact
solution
dr =
7.6560 3.5360 1.6560 6.2560 7.0160 -2.0640 -22.6640 -52.1440 -83.5440
-105.5840 -112.5840
```

```
dy =
20.000 0.1600 0.7200 7.0400 8.8000 0 -21.0400 -51.6800 -84.9600
-109.6000 -110.0000
```

2.6.2 INTEGRATION

2.6.2.1 Definite Integral

A definite integral can be approximated by a weighted sum of function values at points within the specified integration interval $[a \ b]$. A numerical integration formula to approximate $\int_a^b f(x)dx$ has the form

$$\int_a^b f(x)dx \approx \sum_{i=0}^n c_i f(x_i)$$

where the parameters c_i depend on the particular numerical method.

The trapezoid rule, one of the simplest methods to estimate a definite integral, approximates $f(x)$ by the straight line that passes through adjacent two points. The trapezoid rule is represented as

$$\int_a^b f(x)dx \approx \frac{h}{2} \{f(a) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(b)\}$$

where $h = (b - a)/n$.

The widely used Simpson's rule approximates the function to be integrated by a quadratic polynomial. The approximate integral by the Simpson's rule is given by

$$\int_a^b f(x)dx \approx \frac{h}{3} \{f(a) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \dots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(b)\}$$

where n is even.

The midpoint rule uses only function evaluations at points within the integral interval. The midpoint rule is given by

$$\int_a^b f(x)dx \approx h \sum_{i=0}^n f(x_i), \quad x_i = a + \left(i - \frac{1}{2}\right)h$$

MATLAB® provides built-in functions that evaluate definite integrals for a given data set.

The *trapz* function implements the Trapezoidal rule. A simple representation of its syntax is

```
z = trapz(x, y)
```

where x and y are vectors of independent and dependent variables, respectively.

The *cumtrapz* function evaluates the cumulative integral. It has the general syntax

```
z = cumtrapz(x, y)
```

where x and y are vectors of independent and dependent variables, respectively.

The *quad* function implements the adaptive Simpson quadrature, and the function *quadl* performs the Lobatto quadrature rule. Simple expressions of their syntax are

```
z = quad(fun, a, b, tol)
z = quadl(fun, a, b, tol)
```

where *z* is the integral of the function *fun* over the ranges from *a* to *b*, and *tol* is the desired absolute error tolerance. If *tol* is not specified, a default value of 1×10^{-6} is used.

Example 2.30: Use of *trapz* and *cumtrapz*

[Table 2.19](#) shows a series of time spot measurements of the velocity of a falling sphere. Determine the distance traveled when $t = 2.5$ sec. Find the cumulative distance traveled at each time spot.

Solution

The distance traveled to a certain time spot is given by the function *trapz*, and the distance traveled at each time spot can be calculated by the function *cumtrapz*. The following script estimates the distances:

```
>> t = [0 0.5 1.2 1.6 2.5 3.1 4.8 6.9]; t1 = [0 0.5 1.2 1.6 2.5];
>> v = [0 5 12 15 23 28 38 47]; v1 = [0 5 12 15 23];
>> z1 = trapz(t1,v1)
z1 =
    29.7000
>> z = cumtrapz(t,v)
z =
    0 1.2500 7.2000 12.6000 29.7000 45.0000 101.1000 190.3500
```

[Table 2.20](#) summarizes the results.

Example 2.31: Use of *quad* and *quadl*

MATLAB® provides a demonstrative function humps defined as

$$f(x) = \frac{1}{(x-q)^2 + 0.01} + \frac{1}{(x-r)^2 + 0.04} + s$$

TABLE 2.19
Velocity of a Falling Mass

<i>t</i> (sec)	0	0.5	1.2	1.6	2.5	3.1	4.8	6.9
<i>v</i> (m/sec ²)	0	5	12	15	23	28	38	47

TABLE 2.20
Distances Traveled

<i>t</i> (sec)	0	0.5	1.2	1.6	2.5	3.1	4.8	6.9
<i>v</i> (m/sec ²)	0	5	12	15	23	28	38	47
Distance (m)	0	1.25	7.2	12.6	29.7	45.0	101.1	190.35

Estimate $\int_a^b f(x)dx$ when $a = 0$, $b = 1$, $q = 0.2$, $r = 0.7$, and $s = 4$. Use built-in functions *quad* and *quadl* and compare the results.

Solution

The following script estimates the integrals:

```
>> format long
>> q = 0.2; r = 0.7; s = 4;
>> hfun = @(x) 1./((x-q).^2+0.01)+1./((x-r).^2+0.04)-s;
>> quad(hfun,0,1)
ans =
    32.912352767792420
>> quadl(hfun,0,1)
ans =
    32.912352455611952
```

2.6.2.2 Multiple Integrals

The MATLAB®'s built-in function *dblquad* implements double integration

$$I = \int_a^b \int_c^d f(x,y) dy dx = \int_a^b \left\{ \int_c^d f(x,y) dy \right\} dx = \int_c^d \left\{ \int_a^b f(x,y) dx \right\} dy$$

A simple expression of the syntax for *dblquad* is

```
z = dblquad(fun, xa, xb, yc, yd, tol)
```

where z is the double integral of the function fun over the ranges from xa to xb and yc to yd . If the desired absolute error tolerance tol is not specified, a default value of 1×10^{-6} is used. Here is an example of how *dblquad* can be used to evaluate the double integral

$$I = \int_0^{0.8} \int_0^{0.8} \frac{1}{1-xy} dy dx$$

```
>> I = dblquad(@(x,y) 1./(1-x.*y), 0, 0.8, 0, 0.8)
I =
    0.7900
```

The built-in function *quad2d* can also be used to evaluate the double integral. This function can be used when the range of one of the two independent variables is a function of the other independent variable. As an example, the double integral

$$\int_{x^2}^{2x^4} \int x^2 y dy dx$$

can be estimated by using *quad2d*:

```
>> I = quad2d(@(x,y) x.^2.*y, 1, 2, @(x) x.^2, @(x) x.^4)
I =
    83.9740
```

The built-in function *triplequad* implements triple integration

$$I = \iiint_a^b \int_c^d \int_g^h f(x, y, z) dz dy dx = \int_a^b \left\{ \int_c^d \left\{ \int_g^h f(x, y, z) dz \right\} dy \right\} dx$$

A simple expression of the syntax for *triplequad* is

```
w = triplequad(fun, xa, xb, yc, yd, ze, zf, tol)
```

where w is the triple integral of the function fun over the ranges from xa to xb, yc to yd, and ze to zf. If the desired absolute error tolerance tol is not specified, a default value of 1×10^{-6} is used. Here is an example of how *triplequad* can be used to compute the triple integral

$$I = \iiint_0^1 \int_0^1 \int_0^1 64xy(1-x)^2 zdz dy dx$$

```
>> I = triplequad(@(x,y,z) 64*x.*y.* (1-x) .^2.*z, 0, 1, 0, 1, 0, 1)
I =
1.3333
```

Example 2.32: Double Integral

Evaluate the double integral

$$I = \iint_0^5 \int_0^7 f(x, y) dx dy$$

for the function

$$f(x, y) = 3xy + x - 1.2x^2 - 3y^2 + 25$$

Solution

```
>> fxy = @(x,y) 3*x.^y+x-1.2*x.^2-3*y.^2+25;
>> z = dblquad(fxy, 0, 7, 0, 5)
z =
3.2878e+04
```

2.7 ORDINARY DIFFERENTIAL EQUATIONS

Differential equations are composed of an unknown function and its derivatives. The equation is called an ordinary differential equation (ODE) when this unknown function involves only one independent variable and partial differential equation (PDE) when more than one independent variable is involved.

2.7.1 INITIAL-VALUE PROBLEMS

The most basic method for finding the approximate solution of a 1st-order ODE is Euler's method. Consider a 1st-order ODE, $(dy/dt) = y' = f(t, y)$, with the initial condition $y(t_0) = y_0$ on an

interval $[t_0, t_n]$. We find the approximate solutions y_1, y_2, \dots, y_n at the points $t_1 = t_0 + h, t_2 = t_0 + 2h, \dots, t_n = t_0 + nh$ by computing

$$y_{i+1} = y_i + hf(t_i, y_i)$$

where $h = (t_n - t_0)/n$. In the midpoint method, the approximate values of solutions are given by

$$y_{i+1} = y_{i-1} + 2hf(t_i, y_i)$$

The well-known Runge–Kutta methods are based on the determination of a sequence of approximations to the change $y_{i+1} - y_i$ followed by a combination of these to compute a new value of y . These methods give results that match the higher-order Taylor series expansions, while an evaluation of derivatives of $f(t, y)$ is not required. The optimal 2nd-order Runge–Kutta method is given by

$$k_1 = hf(t_i, y_i)$$

$$k_2 = hf\left(t_i + \frac{2}{3}h, y_i + \frac{2}{3}k_1\right)$$

$$y_{i+1} = y_i + \frac{1}{4}(k_1 + 3k_2)$$

and the classic 4th-order Runge–Kutta method is given by

$$k_1 = hf(t_i, y_i)$$

$$k_2 = hf\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1\right)$$

$$k_3 = hf\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2\right)$$

$$k_4 = hf(t_i + h, y_i + k_3)$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

An n th-order ODE can be transformed into a set of 1st-order ODEs. The general form of an n th-order differential equation can be expressed as

$$f\left(t, y, \frac{dy}{dt}, \frac{d^2y}{dt^2}, \dots, \frac{d^{n-1}y}{dt^{n-1}}\right) = f\left(t, y, y', y'', \dots, y^{(n-1)}\right) = \frac{d^n y}{dt^n} = y^{(n)}$$

The state function can be defined as

$$s(t) = \begin{bmatrix} y(t) \\ y'(t) \\ y''(t) \\ \vdots \\ y^{(n-1)}(t) \end{bmatrix} = \begin{bmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \\ \vdots \\ s_n(t) \end{bmatrix}$$

TABLE 2.21
Some of MATLAB®'s Built-In Functions for Solving ODE

Functions	Solution Method
<i>ode23</i>	Runge–Kutta lower order (2nd and 3rd order)
<i>ode45</i>	Runge–Kutta higher order (4th and 5th order)
<i>ode113</i>	Adams–Bashforth–Moulton

If we substitute these relations into the n th-order differential equation, we have the equivalent system of simultaneous ordinary differential equations, which can be represented by

$$\frac{ds(t)}{dt} = \begin{bmatrix} y'(t) \\ y''(t) \\ y^{(3)}(t) \\ \vdots \\ y^{(n)}(t) \end{bmatrix} = \begin{bmatrix} y'(t) \\ y''(t) \\ y^{(3)}(t) \\ \vdots \\ f(t, y, y', \dots, y^{(n-1)}) \end{bmatrix} = \begin{bmatrix} s_2(t) \\ s_3(t) \\ s_4(t) \\ \vdots \\ f(t, s_1(t), s_2(t), \dots, s_n(t)) \end{bmatrix}$$

The solution of this system requires that n initial conditions be known at the starting value of t .

MATLAB® provides many built-in functions for solving ordinary differential equations. In order to use them, the given differential equation should be transformed into the equivalent system of 1st-order differential equations. [Table 2.21](#) presents some of the built-in functions.

MATLAB's built-in functions for solving ODEs can be called in a number of different ways. A simple representation of the syntax is

```
[t, y] = solver(odefun,tspan,y0,options)
```

where solver is the name of MATLAB's built-in function, odefun is the name of the function defining the system of 1st-order ODEs to be solved, tspan is the span of independent variable, y0 is the vector of initial values, and options is a data structure created with the *odeset* function to control features of the solution.

2.7.2 STIFF SYSTEM

A stiff system involves both rapidly changing components and the slowly changing ones. In some cases, the rapidly varying components die away quickly and the solution becomes dominated by the slowly varying components.

MATLAB® provides a number of built-in functions for solving stiff systems of ODEs. [Table 2.22](#) shows some of these solvers.

TABLE 2.22
Some of MATLAB®'s Built-In Functions for Stiff Systems

Functions	Solution Method
<i>ode23s</i>	Modified 2nd-order Rosenbrock
<i>ode23t</i>	Trapezoidal rule
<i>ode23tb</i>	Implicit Runge–Kutta method
<i>ode15s</i>	Variable-order implicit multistep method

Example 2.33: Solution of an ODE

Solve

$$\frac{dy}{dt} = e^{-t}, \quad y(0) = -1$$

from $t = 0$ to 1.

Solution

A function can be created to hold the given differential equation:

```
function dy = expfn(t,y)
dy = exp(-t);
end
```

The following script employs the built-in *ode45* function to integrate *expfn* and generate a plot of the solution as shown in [Figure 2.12](#):

```
>> tspan = [0 1]; y0 = -1;
>> [t y] = ode45(@expfn, tspan, y0);
>> plot(t,y), grid, xlabel('t'), ylabel('y(t)')
```

Example 2.34: van der Pol Equation

The van der Pol equation can be expressed as

$$\frac{d^2y_1}{dt^2} - \mu(1 - y_1^2) \frac{dy_1}{dt} + y_1 = 0$$

This equation can be transformed into a set of 1st-order differential equations as follows:

$$\frac{dy_1}{dt} = y_2$$

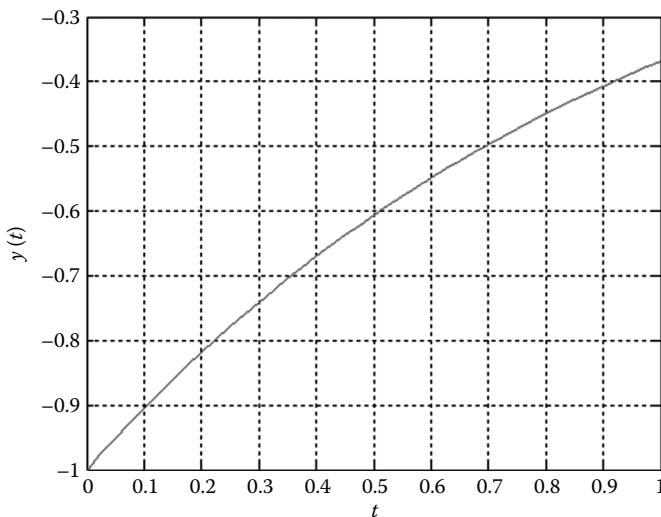


FIGURE 2.12 Plot of the solution of $(dy/dt) = e^{-t}, y(0) = -1$.

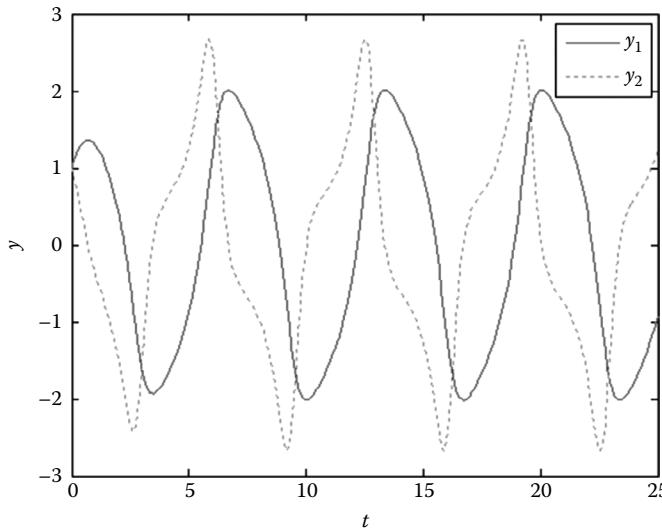


FIGURE 2.13 Plot of the solution of the van der Pol equation.

$$\frac{dy_2}{dt} = -y_1 + \mu(1 - y_1^2)y_2$$

Plot the changes of y_1 and y_2 with respect to time t from $t = 0$ to 25. $\mu = 1$ and the initial conditions are $y_1(0) = y_2(0) = 1$.

Solution

First, create a function to hold the van der Pol equation:

```
function dy = vdpeqn(t,y, mu)
% y(1) = y1, y(2) = y2
dy = [y(2); -y(1) + mu*(1-y(1)^2)*y(2)];
end
```

The following script employs the built-in *ode45* function to integrate *vdpeqn* and generate a plot of the solution as shown in Figure 2.13:

```
>> mu = 1; tspan = [0 25]; y0 = [1 1];
>> [t y] = ode45(@vdpeqn, tspan, y0, [], mu);
>> plot(t,y(:,1),t,y(:,2),':'), legend('y_1','y_2'), xlabel('t'), ylabel('y')
```

Example 2.35: Well-Mixed Tanks

Figure 2.14 shows a series of three well-mixed tanks. From mass balance equations, we have

$$\frac{dV_1}{dt} = q_0 + m - q_1, \quad V_1 \frac{dx_1}{dt} = q_0(x_0 - x_1) - mx_1, \quad V_2 \frac{dx_2}{dt} = q_1(x_1 - x_2), \quad V_3 \frac{dx_3}{dt} = q_2(x_2 - x_3)$$

where x_i ($i = 1, 2, 3$) is the concentration (mol/liter) of the solution contained in the tank i . Under normal steady-state operation, m is maintained at 0 and q_i ($i = 1, 2, 3$) is kept constant. At a certain time ($t = 0$), m is suddenly increased to 12 liter/min. Plot the concentrations in the three tanks from $t = 0$ to 2. The initial conditions are $x_0 = 0.15$ mol/liter and $q_0 = 15$ liter/min, and the initial volume of each tank is 20 liter.

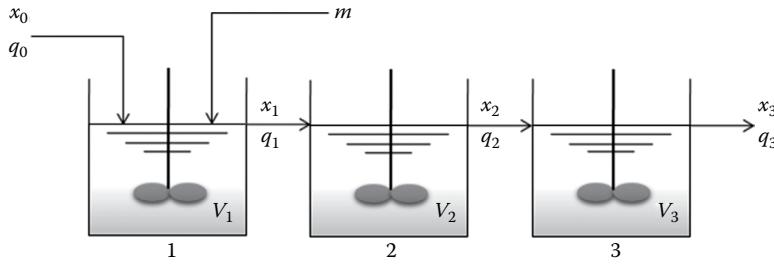


FIGURE 2.14 A series of three well-mixed tanks.

Solution

When $m = 0$, $V_1 = V_2 = V_3 = 20$ liter and q_i ($i = 1, 2, 3$) is constant ($q_i = 15$ liter/min). Thus, V_2 and V_3 are constant and independent with changes in m . Substituting the given data, we have

$$\frac{dx_1}{dt} = \frac{2.25 - 27x_1}{V_1}, \quad \frac{dx_2}{dt} = \frac{15}{20}(x_1 - x_2), \quad \frac{dx_3}{dt} = \frac{15}{20}(x_2 - x_3), \quad \frac{dV_1}{dt} = 12$$

The system of differential equations can be defined as a function file where $y_1 = x_1$, $y_2 = x_2$, $y_3 = x_3$, and $y_4 = V_1$:

```
function dy = cstm(t,y)
dy = [(2.25-27*y(1))./y(4);
      15*(y(1)-y(2))/20;
      15*(y(2)-y(3))/20;
      12];
end
```

The following script employs the *ode45* solver to integrate *cstm* and generate a plot of the solution as shown in [Figure 2.15](#):

```
>> tspan = [0 2]; y0 = [0.15 0.15 0.15 20];
>> [t y] = ode45(@cstm, tspan, y0);
```

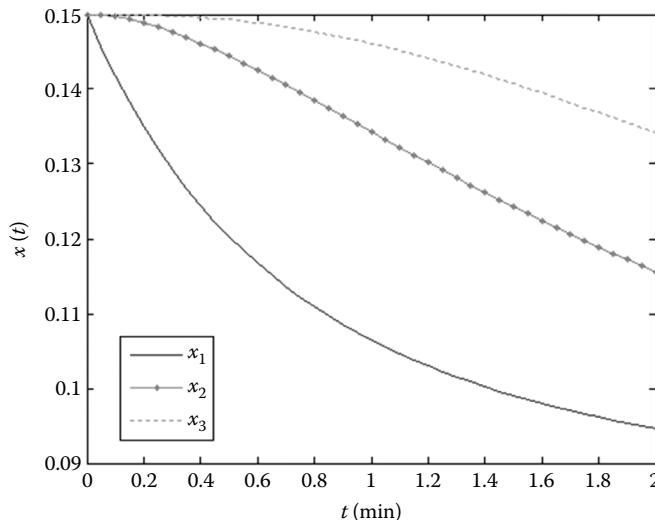


FIGURE 2.15 Concentration changes in mixed tanks.

```
>> plot(t,y(:,1),t,y(:,2),'.-',t,y(:,3),':'), xlabel('t(min)'),  
ylabel('x(t)')  
>> legend('x_1', 'x_2', 'x_3')
```

Example 2.36: Stiff Differential Equation⁵

A biological process involving the growth of a biomass from substrate can be represented as

$$\frac{dB}{dt} = \frac{kBS}{K+S}, \quad \frac{dS}{dt} = -\frac{0.75kBS}{K+S}$$

where B and S are the biomass and substrate concentrations, respectively. Solve these differential equations from $t = 0$ to 20. At $t = 0$, S and B are 5 and 0.05, respectively. The reaction kinetics are $k = 0.3$ and $K = 0.000001$.

Solution

This system is a typical stiff system and we can use the built-in functions for stiff equations. A function can be created to hold the given differential equations:

```
function dy = biom(t,y,k,K)
% y(1): B, y(2): S
dy = [k*y(1)*y(2)/(K + y(2));
      -0.75*k*y(1)*y(2)/(K + y(2))];
end
```

Since the concentration of the substrate cannot be zero, we provide reduced error tolerance (10^{-8}) to the solver as an option. The following script employs the *ode15s* solver to integrate *biom* and generate a plot of the solution. We can use other solvers such as *ode23s*, *ode23t*, or *ode23tb* to solve the stiff system. The following code generates the plot of change of substrate and biomass concentrations as shown in Figure 2.16

```
>> tspan = [0 20]; y0 = [0.05 5]; k = 0.3; K = 1e-6;
>> options = odeset('RelTol',1e-8,'AbsTol',[1e-8 1e-8]);
>> [t y] = ode15s(@biom, tspan, y0, options,k,K);
>> plot(t,y(:,1),t,y(:,2),':'), xlabel('t(min)'), ylabel('y(t)')
>> legend('B(t)', 'S(t)', 'location', 'best')
```

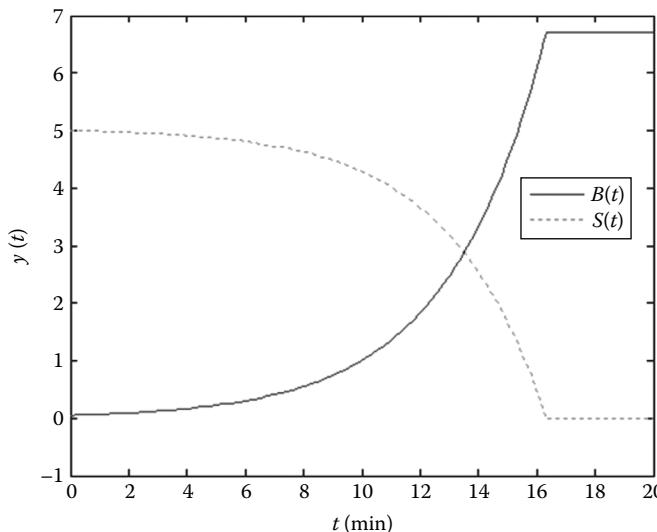


FIGURE 2.16 Change of substrate and biomass concentrations.

Example 2.37: Fluidized Packed Bed Catalytic Reactor⁶

The irreversible gas phase catalytic reaction



is to be carried out in a fluidized packed bed reactor. Material and energy balances for this reactor yield

$$\frac{dP}{d\tau} = P_e - P + H_g (P_p - P)$$

$$\frac{dT}{d\tau} = T_e - T + H_T (T_p - T) + H_W (T_w - T)$$

$$\frac{dP_p}{d\tau} = \frac{H_g}{A} \{P - P_p(1+K)\}$$

$$\frac{dT_p}{d\tau} = \frac{H_T}{C} \{(T - T_p) + F K P_p\}$$

$$K = 6 \times 10^{-4} \exp\left(20.7 - \frac{1000}{T_p}\right)$$

where

T (°R) is the temperature of the reactant

P (atm) is the partial pressure of the reactant

T_p (°R) is the temperature of the reactant at the surface of the catalyst

P_p (atm) is the partial pressure of the reactant at the surface of the catalyst

K is the rate constant (dimensionless)

τ is time (dimensionless)

and the subscript e is the inlet condition

The parameters and constants used in the model equations are

$$H_g = 320, \quad T_e = 600, \quad H_T = 266.67, \quad H_W = 1.6, \quad T_w = 720, \quad F = 8000, \quad A = 0.17142, \\ C = 205.74, \quad P_e = 0.1$$

Solve the differential equations from $\tau = 0$ to 1500 and plot the changes of dependent variables. Initial conditions are $P(0) = 0.1$, $T(0) = 600$, $P_p = 0$, and $T_p = 761$.

Solution

First, create the function to hold differential equations:

```
function dy = catfb(t,y,g)
% y(1): P, y(2): T, y(3): Pp, y(4): Tp
A = g.A; C = g.C; F = g.F; Hg = g.Hg; Ht = g.Ht; Hw = g.Hw;
Pe = g.Pe; Te = g.Te; Tw = g.Tw;
K = 6e-4 * exp(20.7 - 15000/y(4));
dy = [Pe - y(1) + Hg*(y(3) - y(1));
      Te - y(2) + Ht*(y(4) - y(2)) + Hw*(Tw - y(2));
      Hg*(y(1) - y(3)*(1 + K))/A;
      Ht*(y(2) - y(4)) + F*K*y(3))/C];
end
```

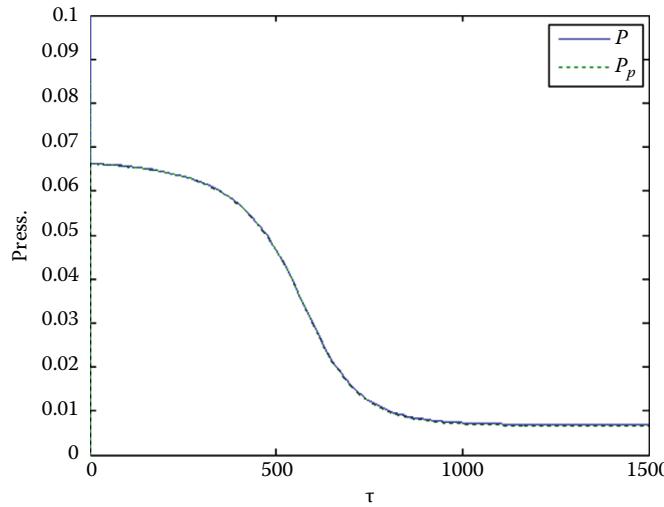


FIGURE 2.17 Change of reaction pressure.

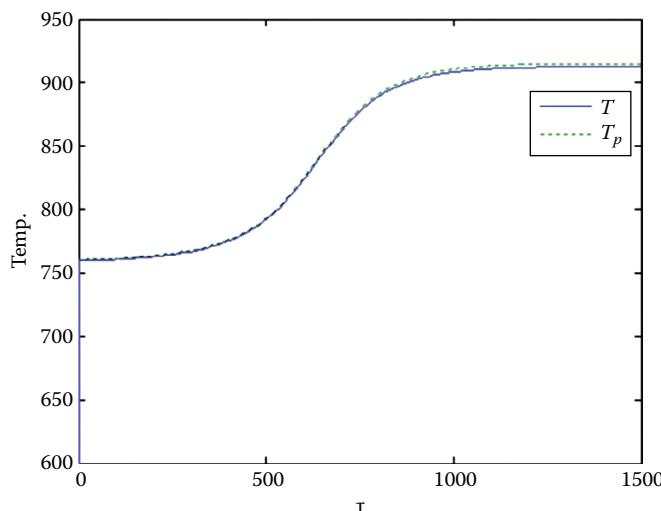


FIGURE 2.18 Change of reaction temperature.

The following script employs the built-in *ode15s* function for stiff systems to integrate *catfb* and generate plots of the solutions as shown in Figures 2.17 and 2.18. The parameters and constants are fed into the function by using the structure:

```

>> g.A=0.17142; g.C=205.74; g.F=8000; g.Hg=320; g.Ht=266.67; g.Hw=1.6;
>> g.Pe=0.1; g.Te=600; g.Tw=720;
>> tspan = [0 1500]; Yinit = [0.1 600 0 761];
>> options = odeset('RelTol',1e-8,'AbsTol',[1e-8 1e-8 1e-8 1e-8]);
>> [t Y] = ode15s(@catfb, tspan, Yinit,options,g);
>> figure(1), plot(t,Y(:,1),t,Y(:,3),':'), xlabel('\tau'), ylabel('Press.'),
    legend('P','P_p')
>> figure(2), plot(t,Y(:,2),t,Y(:,4),':'), xlabel('\tau'), ylabel('Temp.'),
    legend('T','T_p')

```

2.7.3 BOUNDARY-VALUE PROBLEMS

Problems where the value of an unknown function, or its derivative, is constrained at two different points are known as boundary-value problems. MATLAB® provides the built-in function *bvp4c* to solve boundary-value problems. *bvp4c* produces a solution that is continuous on $[a, b]$ and has a continuous first derivative there. A simple expression of the syntax for *bvp4c* is

```
sol = bvp4c(odefun,bcfun,solinit)
```

where *odefun* is a function handle that evaluates the differential equations, *bcfun* is a function handle that computes the residual in the boundary conditions, and *solinit* is a structure containing the initial guess for a solution. The procedure to use *bvp4c* consists of four steps:

Step 1: Transform the set of differential equations into a system of 1st-order differential equations, and create a function to hold the equation system (let's call it *bcprob.m*). It can have the form

```
dy = bcprob(x,y)
```

For a scalar *x* and a column vector *y*, *bcprob*(*x,y*) returns a column vector, *dy*, representing $f(x,y)$.

Step 2: Arrange the boundary conditions in the form of $f(y(a),y(b)) = 0$ and save it as a function (let's call it *bcval.m*). For two-point boundary-value conditions of the form $bc(y(a),y(b))$, *bcfun* can have the form

```
res = bcval(ya,yb)
```

where *ya* and *yb* are column vectors corresponding to *y(a)* and *y(b)*. The output *res* is a column vector.

Step 3: Initialize the value of the dependent variable *y*. We assume *y* values at several points in $[a b]$ (let's call it *initsol*).

Step 4: Using the functions and initial values, call the built-in function *bvp4c*. The function *bvp4c* returns a structure (let's call it *result*), which has several fields as shown in [Table 2.23](#).

As an example, we now solve the boundary-value problem defined by

$$y'' + |y| = 0, \quad y(0) = 0, \quad y(4) = -2$$

by using the built-in function *bvp4c*.⁷

Step 1: We need to rewrite the differential equation as a system of two 1st-order ordinary differential equations:

$$z'_1 = y' = z_2, \quad z'_2 = y'' = -|y| = -|z_1|$$

TABLE 2.23
Fields of the Structure Generated by *bvp4c*

Field	Description
result.x	Values of the independent variables selected by <i>bvp4c</i>
result.y	Approximation to $y(x)$ at the mesh points of <i>x</i>
result.yp	Approximation to $y'(x)$ at the mesh points of <i>x</i>
result.stats	Computational cost statistics
result.solver	<i>bvp4c</i>

where $z_1 = y$ and $z_2 = y'$. Create a function to hold these equations (let's call it *zode.m*):

```
function dz = zode(x,z)
dz = [z(2) ; -abs(z(1))];
```

Step 2: The boundary conditions can be rearranged as $y(0) = 0$, $y(4) + 2 = 0$ or $z_1(0) = 0$, $z_1(4) + 2 = 0$. Create a function to hold these conditions (let's call it *zobc.m*):

```
function res = zobc(za,zb)
res = [za(1) ; zb(1)+2];
```

Step 3: The range of the independent variable x is $x = [a \ b] = [0 \ 4]$. Divide this range into 10 subintervals and initialize the values of dependent variable y at these points. Let $z_1(x) = y(x) = 1$ and $z_2(x) = y'(x) = 0$ (i.e., $[z_1 \ z_2] = [1 \ 0]$). These conditions can be fed into the function *bvpinit* as

```
zinit = bvpinit(linspace(0,4,10), [1 0]);
```

Step 4: Solve the problem using *bvp4c*:

```
sol = bvp4c(@zode,@zobc,zinit);
```

The structure *sol* returned by *bvp4c* has the following fields:

```
sol =
  solver: 'bvp4c'
  x: [1x20 double]
  y: [2x20 double]
  yp: [2x20 double]
  stats: [1x1 struct]
```

We can evaluate the numerical solution at 100 equally spaced points and plot $y(x)$ as

```
x=linspace(0,4,100); y=deval(sol,x); plot(x,y(1,:)), xlabel('x'), ylabel('y')
```

The plot of solutions of two-point BVP is shown in [Figure 2.19](#).

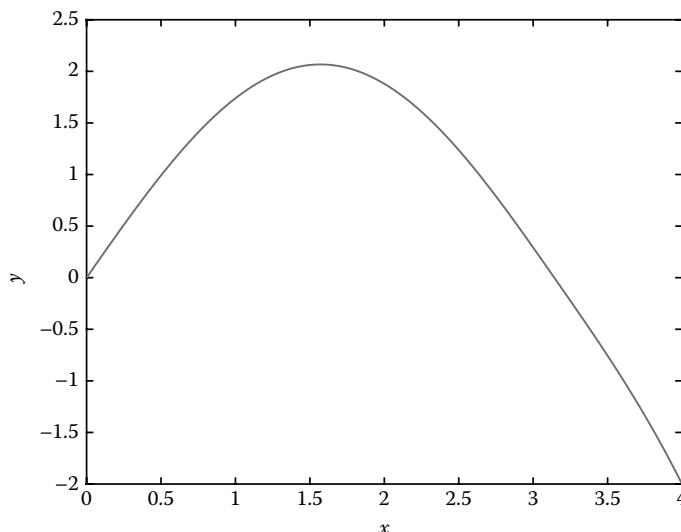


FIGURE 2.19 Plot of solutions of a two-point BVP.

Example 2.38: Boundary-Value Problem

Solve the equation

$$x^2 \frac{d^2y}{dx^2} - 6y = 0$$

The range of x is $1 \leq x \leq 2$, and the boundary conditions are $y(1) = 1$, $y(2) = 1$.

Solution

The differential equation can be rearranged as

$$\frac{dz_1}{dx} = z_2, \quad \frac{dz_2}{dx} = \frac{6z_1}{x^2}$$

where $z_1 = y$ and $z_2 = dy/dx$. We can see that $z_1(1) = y(1) = 1$, $z_1(2) = y(2) = 1$.

Step 1: Create a function that holds differential equations $z'_1 = z_2$ and $z'_2 = 6z_1/x$ (let's call it *bcprob*).

```
function deset = bcprob(x,z)
deset = [z(2); 6*z(1)./x.^2];
end
```

Step 2: The boundary conditions $y(1) = 1$ and $y(2) = 1$ can be rewritten as $y(1) - 1 = 0$ and $y(2) - 1 = 0$ or $z_1(1) - 1 = 0$ and $z_1(2) - 1 = 0$. Denote the boundary condition at $a = 1$ as $z_1(1) = z_1(a) = za$ and the condition at $b = 2$ as $z_1(2) = z_1(b) = zb$. We can create the function that holds these boundary conditions as

```
function bcset = bcval(za,zb)
bcset = [za(1)-1; zb(1)-1];
end
```

Step 3: Suppose that the interval $[a \ b] = [1 \ 2]$ is divided into 10 subintervals and that the value of the dependent variable y is constant as 1 at all subinterval points. Since $z_1(x) = 1$ and $z_2(x) = z'_1 = 0$ at the interval $[1 \ 2]$, the initial guesses can be written as $[z_1(x) \ z_2(x)] = [1 \ 0]$:

```
>> initsol = bvpinit(linspace(1,2,10), [1 0]);
```

Step 4: Solve the equation by calling the function *bvp4c* as

```
>> res = bvp4c(@bcprob, @bcval, initsol);
```

The following scripts show the procedure and generate the plot shown in [Figure 2.20](#):

```
>> initsol = bvpinit(linspace(1,2,10), [1 0]);
>> res = bvp4c(@bcprob, @bcval, initsol);
>> x = linspace(1,2); y = deval(res,x);
>> plot(x,y(1,:)), xlabel('x'), ylabel('y')
```

deval(res,x) evaluates values of the solution structure *res* at *x*. The structure *res* returned by *bvp4c* has the following fields:

```
>> res
res =
solver: 'bvp4c'
x: [1 1.1111 1.2222 1.3333 1.4444 1.5556 1.6667 1.7778 1.8889 2]
y: [2x10 double]
yp: [2x10 double]
stats: [1x1 struct]
```

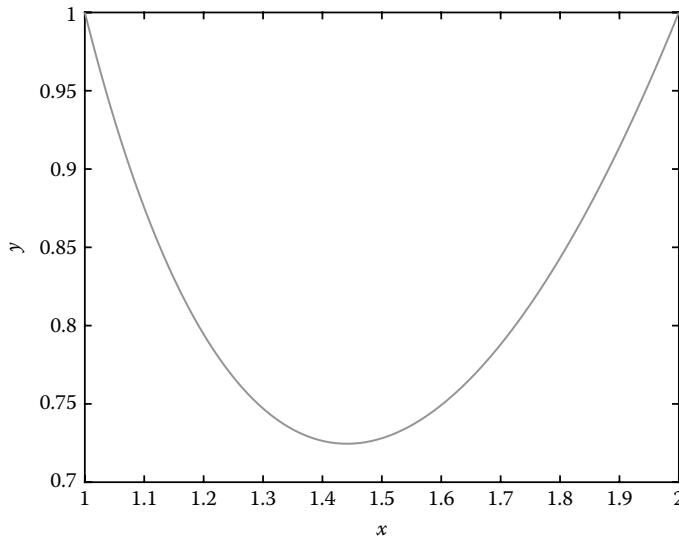


FIGURE 2.20 Solution of a BVP by *bvp4c*.

2.8 PARTIAL DIFFERENTIAL EQUATIONS

2.8.1 CLASSIFICATION OF PARTIAL DIFFERENTIAL EQUATIONS

Partial differential equations (PDEs) are classified according to their order, linearity, and boundary conditions.

2.8.1.1 Classification by Order

The order of a partial differential equation is determined by the highest-order partial derivative in that equation. Examples of 1st-, 2nd-, and 3rd-order partial differential equations are

$$\text{1st-order: } \frac{\partial u}{\partial x} - \alpha \frac{\partial u}{\partial y} = 0$$

$$\text{2nd-order: } \frac{\partial^2 u}{\partial x^2} + u \frac{\partial u}{\partial y} = 0$$

$$\text{3rd-order: } \left(\frac{\partial^3 u}{\partial x^3} \right)^2 + \frac{\partial^2 u}{\partial x \partial y} + \frac{\partial u}{\partial y} = 0$$

2.8.1.2 Classification by Linearity

Partial differential equations can be classified into linear, quasilinear, and nonlinear equations. Let's consider the 2nd-order partial differential equation

$$a(\cdot) \frac{\partial^2 u}{\partial y^2} + 2b(\cdot) \frac{\partial^2 u}{\partial x \partial y} + c(\cdot) \frac{\partial^2 u}{\partial x^2} + d(\cdot) = 0$$

Linear: The coefficients are constants or functions of the independent variables only.

Quasilinear: The coefficients are functions of the dependent variable.

Nonlinear: The coefficients are functions of derivatives of the same order as that of the equation.

2.8.1.3 Classification of Linear 2nd-Order Partial Differential Equations

The general form of partial differential equations we consider is

$$a \frac{\partial^2 u}{\partial y^2} + 2b \frac{\partial^2 u}{\partial x \partial y} + c \frac{\partial^2 u}{\partial x^2} + d \frac{\partial u}{\partial x} + e \frac{\partial u}{\partial y} + fu + g = 0$$

Elliptic: $b^2 - ac < 0$ (example: Laplace's equation $(\partial^2 u / \partial x^2) + (\partial^2 u / \partial y^2) = 0$)

Parabolic: $b^2 - ac = 0$ (example: heat conduction, or diffusion equation $\alpha(\partial^2 u / \partial x^2) = (\partial u / \partial t)$)

Hyperbolic: $b^2 - ac > 0$ (example: wave equation $\alpha^2(\partial^2 u / \partial x^2) = (\partial^2 u / \partial t^2)$)

2.8.1.4 Classification by Initial and Boundary Conditions

The initial and boundary conditions associated with the partial differential equations must be specified in order to obtain unique solutions to these equations. In general, the boundary conditions for partial differential equations can be divided into three categories. Let's consider the one-dimensional unsteady-state heat conduction equation

$$\alpha \frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t}$$

Dirichlet conditions (1st kind): The values of the dependent variable are given at fixed values of the independent variable. Examples of Dirichlet conditions for the heat conduction equation are

$$T = f(x) \quad (t = 0, 0 \leq x \leq 1)$$

or

$$T = T_0 \quad (t = 0, 0 \leq x \leq 1)$$

Boundary conditions of the 1st kind are represented as

$$T = f(x) \quad (x = 0, t > 0), \quad T = T_1 \quad (x = 1, t > 0)$$

Neumann conditions (2nd kind): The derivative of the dependent variable is given as a constant or as a function of the independent variable. For example,

$$\frac{\partial T}{\partial x} = 0 \quad (x = 1, t \geq 0)$$

Cauchy conditions: These conditions are combinations of Dirichlet and Neumann conditions.

Robbins conditions (3rd kind): The derivative of the dependent variable is given as a function of the dependent variable itself. For example,

$$k \frac{\partial T}{\partial x} = h(T - T_f) \quad (x = 0, t \geq 0)$$

where h is the heat transfer coefficient of the fluid being considered.

2.8.2 SOLUTION OF 1ST-ORDER PARABOLIC/ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS USING *pdepe*

The MATLAB® built-in function *pdepe* solves initial-boundary-value problems for parabolic/elliptic partial differential equations of the form⁸

$$g\left(x, t, u, \frac{\partial u}{\partial x}\right) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left(x^m f\left(x, t, u, \frac{\partial u}{\partial x}\right) \right) + r\left(x, t, u, \frac{\partial u}{\partial x}\right)$$

where

- $f(x, t, u, (\partial u / \partial x))$ is a flux term
- $r(x, t, u, (\partial u / \partial x))$ is a source term
- $g(x, t, u, (\partial u / \partial x))$ is a diagonal matrix

The diagonal elements of this matrix are either identically zero or positive. An element that is identically zero corresponds to an elliptic equation and otherwise to a parabolic equation. m can be 0, 1, or 2, corresponding to slab, cylindrical, or spherical symmetry, respectively. If $m > 0$, then a must be greater than zero. The partial differential equations hold for $t_0 \leq t \leq t_f$ and $a \leq x \leq b$. The interval $[a, b]$ must be finite.

The initial condition at $t = t_0$ is represented as

$$u(x, t_0) = u_0(x)$$

and the boundary condition at $x = a$ or $x = b$ is given by

$$p(x, t, u) + q(x, t) f\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0$$

A simple expression of the syntax for *pdepe* is

```
sol = pdepe(m, pdefun, icfun, bcfun, xmesh, tspan)
```

where m is a parameter corresponding to the symmetry of the problem, *pdefun* is a handle to a function that defines the components of the partial differential equation, *icfun* is a handle to a function that defines the initial conditions, *bcfun* is a handle to a function that defines the boundary conditions, and *xmesh* is a vector specifying the points at which a numerical solution is requested for every value in *tspan*. *tspan* is a vector specifying the points at which a solution is requested for every value in *xmesh*. *pdepe* returns values of the solution (*sol*) on a mesh provided in *xmesh*.

The procedure to use *pdepe* consists of four steps:

- Step 1: Specify the system of partial differential equations. Define m , g , f , and r using the given data and conditions, and create a function to hold these functions (let's call it *pdeeq.m*).
- Step 2: Specify initial conditions. Define $u(x, t_0) = u_0(x)$ at $t = t_0$ and create a function to hold it (let's call it *pdeic.m*).
- Step 3: Set boundary conditions. At $x = a$ or $x = b$, specify p , q , and f in the equation $p(x, t, u) + q(x, t) f(x, t, u, (\partial u / \partial x)) = 0$ and create a function to hold these functions (let's call it *pdebc.m*).
- Step 4: Solve the partial differential equation using *pdepe*:

```
sol = pdepe(m, @pdeeq, @pdeic, @pdebc, x, t)
```

Example 2.39: One-Dimensional Parabolic PDE

The temperature $u(x, t)$ in a wall of unit length can be described by the one-dimensional heat equation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

The thickness of the wall is 1 m and the initial profile of the temperature in the wall at $t = 0$ sec is uniform at $T = 90^\circ\text{C}$. At time $t = 0$, the ambient temperature is suddenly changed to 15°C and

held there. If we assume that there is no convection resistance, the temperature of both sides of the wall is also held constant at 15°C. Determine the temperature distribution graphically within the wall from $t = 0$ to $t = 21,600$ sec. The wall property can be assumed as $\alpha = 4.8 \times 10^{-7}$ m/sec².

Solution

Step 1: From the heat equation, we have $((1/\alpha)(\partial u/\partial t)) = (\partial^2 u/\partial x^2)$. Thus, we can see that $m = 0$, $g = 1/\alpha$, $f = \partial u/\partial x$, and $r = 0$. Create a function that holds g , f , and r :

```
function [g,f,r] = pdeTde(x,t,u,DuDx)
alpha = 4.8e-7;
g = 1/alpha;
f = DuDx;
r = 0;
end
```

Step 2: Set the initial conditions. At $t = 0$, $u(x, t_0) = u(x, 0) = u_0(x) = 90$. We can write a function that defines initial conditions as

```
function u0 = pdeTic(x)
u0 = 90;
end
```

Step 3: Specify the boundary conditions $p(x, t, u) + q(x, t)f(x, t, u, (\partial u/\partial x)) = 0$. Since the temperature at both sides of the wall is 15°C, $p = pl = ul - 15$ and $q = ql = 0$ at $x = 0$, and $p = pr = ur - 15$ and $q = qr = 0$ at $x = 1$. Create a function that holds these boundary conditions.

```
function [pl,ql,pr,qr] = pdeTbc(xl,ul,xr,ur,t)
pl = ul-15;
ql = 0;
pr = ur-15;
qr = 0;
end
```

Step 4: Set the intervals for x and t , and call the function *pdepe* to solve the equation.

Suppose that the range of x is divided into 20 subintervals and the time span is divided into 180 subintervals. The following script generates the temperature profile in the wall as shown in [Figure 2.21](#):

```
>> m = 0; x = linspace(0,1,20);
>> t = linspace(0,21600,54);
```

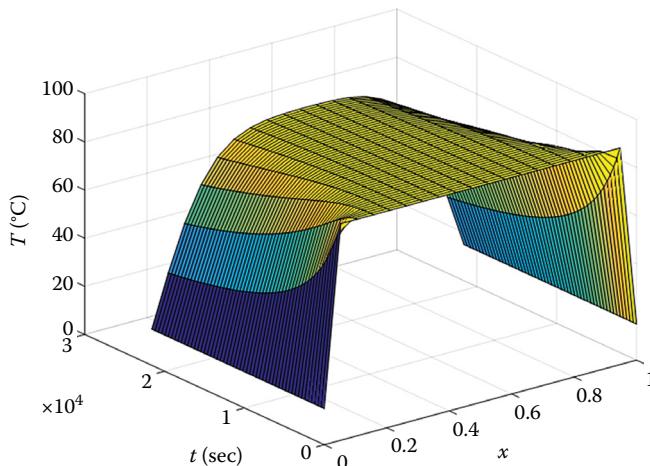


FIGURE 2.21 One-dimensional temperature profile.

```
>> sol = pdepe(m,@pdeTde,@pdeTic,@pdeTbc,x,t);
>> T = sol(:,:,1); surf(x,t,T)
>> xlabel('x'), ylabel('t(sec)'), zlabel('T(deg C)')
```

2.9 HISTORICAL DEVELOPMENT OF PROCESS ENGINEERING SOFTWARE

The early development of chemical process engineering as a discipline was largely experimental and empirical. In typical process operations, the interaction of process variables with one another, and their effect on the performance of the operation, was not fully understood. Therefore, the approach adopted was to perform a number of experiments with the variables covering a wide operation range and to correlate the data using certain statistical methods and dimensional analyses. The resulting empirical model could be used for the design and analysis of chemical process systems.

Alongside the development of chemical process engineering, the use of computer power has been exploited by many process engineers. The history of simulations using computers started in 1946 with the first general-purpose computer (ENIAC). The first deployment of this computer was managed by John von Neumann to model the process of nuclear detonation during the Manhattan project.⁹ Initial attempts toward computer simulation of chemical process engineering systems were made in the early 1950s, and one of the most significant advances in digital simulation took place in the late 1950s with the development of the FORTRAN programming language. The earliest process engineering programs were mostly automated versions of standard chemical engineering textbook modeling procedures for the common unit operations including reactor, distillation, heat exchanger, etc., using the empirical correlations. Derivation of chemical process models took considerable skill and time to develop into computer programs.

By the end of the 1950s, computer science was emerging as a promising field providing new simulation tools. However, up to the 1960s, most chemical engineers lacked the tools to design and simulate a complete process. In 1964, the digital computer program PACER, one of the ancestors of today's chemical process simulators, received its first practical test. It was written in FORTRAN for an IBM 7090 computer and consists of more than 1000 statements. Also in 1964, Imperial College in London released SPEEDUP (Simulation Program for the Economic Evaluation and Design of Unsteady-State Processes), the first process simulator based on the equation-oriented approach. For most chemical process engineers, these programs remained a "black box" model, and the continued use of manual calculations persisted even toward the end of the 1960s. In fact, early generation computers were not adequate to solve some large and sophisticated chemical engineering problems both by virtue of their limited memory and slow speed. In some cases, drastic simplifying assumptions had to be made in order to reduce the complexity of the problem, and this tended to give a distorted picture of the system behavior.¹⁰ In 1966, Simulation Sciences commercialized a computer program for simulating distillation columns, which was the core of the flowsheeting package PROCESS. In 1969, ChemShare released DESIGN, which is a flowsheeting program for gas and oil applications.

During the 1970s, large high-speed digital computers became available for solving complex chemical engineering problems with a considerable increase in the scope and usability of many computer programs. In the early 1970s, integrated process flowsheet package programs like PACER, Monsanto's FLOWTRAN, Kellogg's Flexible Flowsheet, and ICI's FLOWPACK appeared. With these programs, chemical engineers were able to model complex flowsheets on the computer and simulate the effect of interactions between various unit operations at different stages of the process. The traditional concept of designing a string of unit operations for a chemical process gave way to a new generation of computer-aided design techniques, where processes are designed as integrated units using the process flowsheeting approach. In 1976, the U.S. Department of Energy and the MIT launched jointly the Aspen project, which eventually wound up with Aspen Plus, one of the most widely used simulators in the world.

In the 1980s, various specialized packages became available, such as DESIGN II, ASPEN, SIMSCI (PROII), HYSYS, and CHEMCAD, which started appearing on chemical engineering desktops.

TABLE 2.24
Process System Engineering Software

Software	Year	Developer	Applications	Type
PACER	1963	Paul T. Shannon		Sequential modular
PROCESS	1966	Simulation Sciences	Steady-state simulation	Sequential modular
DESIGN	1969	ChemShare	Gas and oil applications	Sequential modular
CHESS	1969	Univ. of Houston	Flowsheeting, sizing, and costing	Sequential modular
FLOWTRAN	1970	Monsanto	Flowsheeting, sizing, and costing	Sequential modular
FLOWPACK II	1972	ICI	Flowsheeting	Sequential modular
DYSCO	1981	Institute of Paper Chemistry	Dynamic simulation	Dynamic modular simulation
ASCEND	1970–1980	Carnegie Mellon Univ.	Dynamic simulation	Equation oriented
Aspen Plus	1976	DOE, MIT	All-purpose flowsheeting	
TISFLO	1970–1980	DSM		Equation oriented
FLOWSIM	Early 1980s	Univ. of Connecticut		Equation oriented
HYSIM HYSYS	Mid-1990s	Hyprotech	Steady-state/dynamic simulation	
SpeedUp	1986	Imperial College	Dynamic simulation	Equation oriented
ProSimPlus	1989	ProSim	Steady-state simulation	Sequential modular
Design II		WinSim	Flowsheeting	
Quasilinear	Late 1980s	CADCentre	Simulation/optimization	Equation oriented
gPROMS		Imperial College, PSE Ltd.	Steady-state/dynamic modeling	Equation oriented
BatchPro Designer		Intelligen	Batch scheduling and design	
COCO simulator		AmsterCHEM	Flowsheeting	Sequential modular

Sources: https://en.wikipedia.org/wiki/List_of_chemical_process_simulators; Martin, M.M., *Introduction to Software for Chemical Engineers*, CRC Press, pp. 291–293, 2015.

The improvement of computer technology allowed better data analysis and understanding of the processes and provided solutions to more complex problems. In 1985, AspenTech released Aspen Plus. During the late 1980s, PRO II was upgraded from PROCESS by Simulation Sciences, and major packages migrated to PCs.⁹

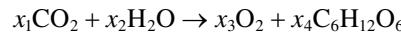
In the early 1990s, all major software packages underwent periodic upgrades, and advanced applications such as pinch analysis were released. In the mid-1990s, major vendors converted the graphical user interface into a central part in the software development, and HYSIM became NYSYS. During this period of time, the process simulation market underwent severe transformations. The few systems that survived include CHEMCAD, Aspen Plus, Aspen HYSYS, PRO/II, ProSimPlus, SuperPro Designer, and gPROMS. Nowadays, most of the object-oriented chemical process simulators are developed based on combination of sequential modular and equation-oriented approaches using languages like C++, C#, MATLAB®, or Java. Table 2.24 summarizes the evolution of commercial process simulators.

For both the chemical engineering student at the university and the chemical process engineer in the chemical industry, increasing emphasis has been placed both on understanding basic physical and chemical principles and on process modeling, and the development of process engineering software has not entirely replaced the early approach of calculating individual unit operations. The practicing chemical engineer will notice that, to some degree, the operation and optimization of chemical plants and the design of minor plant modifications are still largely empirical, backed by operating experience. By the skillful combination of the analytical aspect and experimental empiricism, a framework can be developed for an economical, quantitative, and scientific approach to technical problems in the process industry. In this context, computer-aided design, process modeling,

and simulation become essential tools for the chemical process engineer. One of the most important requirements of a process engineer is not only the ability to use the library of process engineering programs but also the capacity to develop one's own program either as a stand-alone program or appended to an existing software to share some of the library programs. A process engineer with this ability may understand and appropriately respond to error or warning messages that appear in using process engineering programs or in performing computer-aided plant operation.

PROBLEMS

- 2.1** In the photosynthesis reaction, water reacts with carbon dioxide to give glucose and oxygen. This reaction can be expressed as



Determine the values of coefficients x_1 , x_2 , x_3 , and x_4 to balance the equation. Is it possible to determine these values? If not, under what conditions can the solutions be found?

- 2.2** Four reactors are connected by pipes where directions of flow are depicted by means of arrows as shown in Figure P2.2.¹³ The flow rate of the key component is given by the volumetric flow rate Q (liter/sec) multiplied by the concentration C (g/liter) of the component. The incoming flow rate is assumed to be equal to the outgoing rate. Using the flow rates given below, calculate the concentration at each reactor:

$$Q_{13} = 75 \text{ liter/sec}, \quad Q_{24} = 20 \text{ liter/sec}, \quad Q_{33} = 60 \text{ liter/sec}, \quad Q_{21} = 25 \text{ liter/sec}, \\ Q_{32} = 45 \text{ liter/sec}, \quad Q_{43} = 30 \text{ liter/sec}$$

- 2.3** Paraxylene, styrene, toluene, and benzene are to be separated with the array of distillation columns shown in Figure P2.3.¹⁴ Determine the molar flow rates ($\text{kg} \cdot \text{mol/min}$) of D_1 , D_2 , B_1 , and B_2 .
- 2.4** Figure P2.4 shows a flat square plate the sides of which are held at constant temperatures (200°C and 500°C). Find the temperatures at inner nodes (i.e., $T_7 - T_9$, $T_{12} - T_{14}$, $T_{17} - T_{19}$ (200)). The temperature at each inner node is assumed to be given by the average of temperatures of adjacent nodes.
- 2.5** The vapor pressure (mmHg) of *n*-pentane (*A*) and *n*-hexane (*B*) can be calculated from the Antoine equation (T : °C)¹⁵:

$$\log P_A = 6.85221 - \frac{1064.63}{T + 232.0}, \quad \log P_B = 6.87776 - \frac{1171.53}{T + 224.366}$$

1. Calculate the bubble point temperature and equilibrium composition associated with a liquid mixture of 10 mol % *n*-pentane and 90 mol % *n*-hexane at 1 atm.

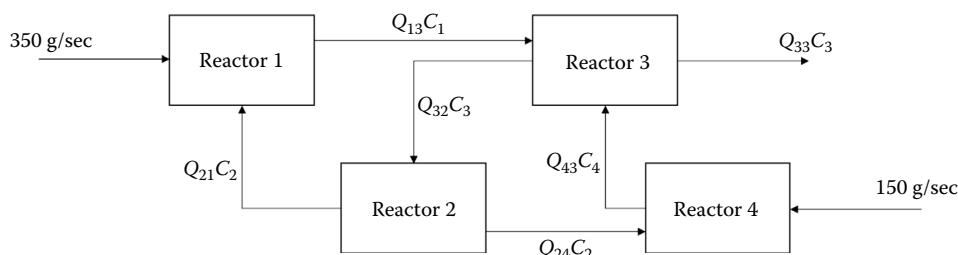


FIGURE P2.2

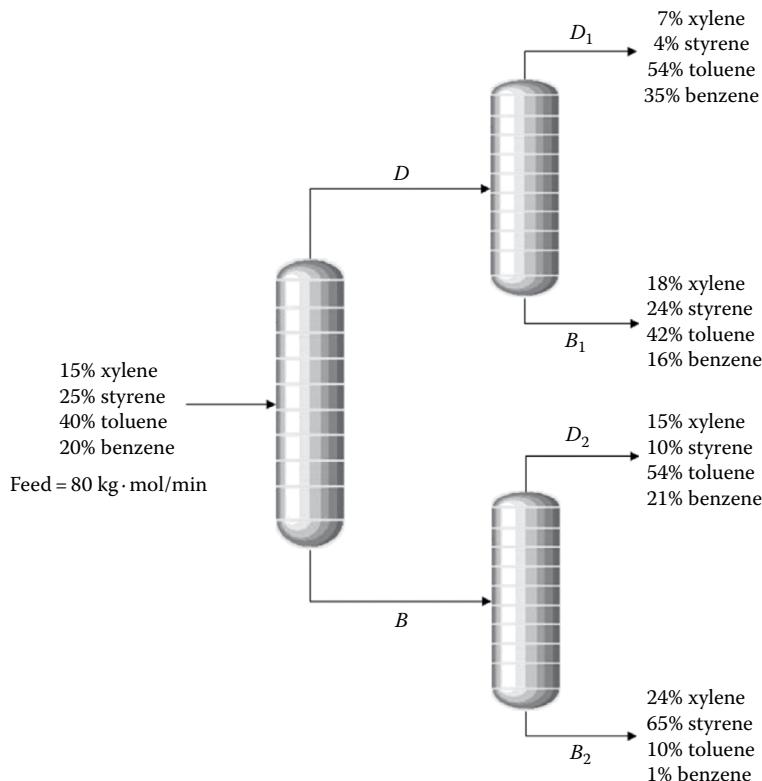


FIGURE P2.3

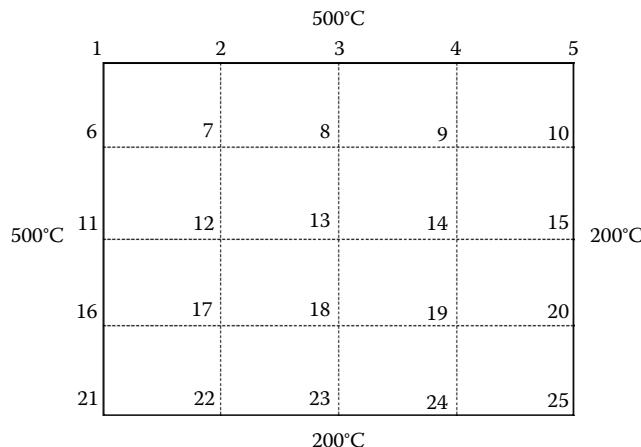


FIGURE P2.4

- Repeat the calculations for liquid mixtures containing 0 mol % up to 100 mol % of *n*-pentane. Plot the bubble point temperature and mol % of *n*-pentane in the vapor phase as a function of the mol % in the liquid phase.
- 2.6** Table P2.6 shows the mole fractions of benzene at a certain temperature and pressure in the benzene–toluene liquid mixture. x and y are the mole fractions of benzene in the liquid and vapor phase, respectively. Find the polynomial to fit these data.

TABLE P2.6
Mole Fraction of Benzene

<i>x</i>	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
<i>y</i>	0.000	0.211	0.378	0.512	0.623	0.714	0.791	0.856	0.911	0.959	1.000

- 2.7** Data of heat capacity C_p (kJ/(kg · mol · K)) versus temperature T (K) for gaseous propane are presented in Table P2.7.¹⁶ Find the polynomial that can be used to represent the variation of C_p as a function of T .
- 2.8** The Margules equation, which describes the activity coefficients of the binary system, can be represented as

$$\ln \gamma_1 = x_2^2 (2B - A) + 2x_2^3 (A - B), \quad \ln \gamma_2 = x_1^2 (2A - B) + 2x_1^3 (B - A)$$

where x_1 and x_2 are the mole fractions of components 1 and 2, respectively. For this specific binary mixture, parameters A and B are constant. The activity coefficients for the binary mixture of benzene (1) and n-heptane (2) are given in Table P2.8. Estimate parameters A and B of the Margules equation using these data.

TABLE P2.7
Heat Capacity of Gaseous Propane

<i>T</i>	C_p	<i>T</i>	C_p
50	34.06	600	128.70
100	41.30	700	142.67
150	48.79	800	154.77
200	56.07	900	163.35
273.15	68.74	1000	174.60
298.15	73.60	1100	182.67
300	73.93	1200	189.74
400	94.01	1300	195.85
500	112.59	1400	201.21

TABLE P2.8
Activity Coefficients of Benzene/n-Heptane Mixture

x_1	γ_1	γ_2
0.0464	1.2968	0.9985
0.0861	1.2798	0.9998
0.2004	1.2358	1.0068
0.2792	1.1988	1.0159
0.3842	1.1598	1.0359
0.4857	1.1196	1.0676
0.5824	1.0838	1.1096
0.6904	1.0538	1.1664
0.7842	1.0311	1.2401
0.8972	1.0078	1.4038

- 2.9** The van Laar equations for the correlation of binary activity coefficients can be expressed as

$$\ln \gamma_1 = \frac{A}{\left(1 + \frac{x_1}{x_2} \frac{A}{B}\right)^2}, \quad \ln \gamma_2 = \frac{B}{\left(1 + \frac{x_2}{x_1} \frac{B}{A}\right)^2}$$

where x_1 and x_2 are the mole fractions of components 1 and 2, respectively. Parameters A and B are constant for this particular binary mixture. The activity coefficients for the binary mixture of benzene (1) and n-heptane (2) are given in [Table P2.8](#). Estimate the parameter values of A and B of the van Laar equation.

- 2.10** [Table P2.10](#) presents data of vapor pressure P (Pa) versus temperature T (K) for benzene.¹⁷ Correlate the data using the Clapeyron equation

$$\log P = A + \frac{B}{T}$$

- 2.11** The Wagner equation can be used to estimate vapor pressure between the triple point and the critical point:

$$\ln P_r = \frac{1}{T_r} \left(a_1 \tau + a_2 \tau^{1.5} + a_3 \tau^3 + a_4 \tau^6 \right)$$

where $T_r = \frac{T}{T_c}$, $P_r = \frac{P}{P_c}$, and $\tau = 1 - T_r$.

[Table P2.11](#) presents ethane vapor pressure (Pa) versus temperature (K).¹⁸ Correlate the data using the Wagner equation. For ethane, $T_c = 305.32$ K and $P_c = 4.872 \times 10^6$ Pa.

- 2.12** Data of thermal conductivity k (W/(m · K)) versus temperature T (K) for gaseous propane are presented in [Table P2.12](#).¹⁹ The nonlinear equation $k = cT^n$ is known to be used to describe the relation between k and T . Estimate parameters c and n that best fit the data.
- 2.13** Data of heat of vaporization of propane ΔH_v (J/kmol) versus temperature T (K) are presented in [Table P2.13](#).²⁰ ΔH_v is known to be represented as a nonlinear function of T as $\Delta H_v = A(T_c - T)^n$ (for propane, $T_c = 369.83$ K). Estimate values of A and n .
- 2.14** Several models were proposed to describe the rates of the reversible catalytic reforming reaction²¹



TABLE P2.10
Vapor Pressure of Benzene

T (K)	P (Pa)
302.39	15,388.0
318.69	30,464.0
330.54	47,571.0
338.94	63,815.0
346.24	81,275.0
353.47	102,040.0
358.87	120,140.0

TABLE P2.11
Vapor Pressure of Ethane

<i>T</i> (K)	$10^{-4}P$ (Pa)	<i>T</i> (K)	$10^{-4}P$ (Pa)
92	0.00017	230	70.0000
100	0.0011	240	96.7000
110	0.0075	250	130.1000
120	0.0350	260	171.2000
130	0.1300	270	221.0000
140	0.3800	276	255.5000
150	0.9700	280	280.6000
158	1.8000	284	307.5000
170	4.3000	288	336.3000
180	7.9000	290	351.4000
192	14.9000	294	383.4000
200	21.7000	298	417.6000
210	33.4000	300	435.6000
220	49.2000	302	454.3000

TABLE P2.12
**Thermal Conductivity
of Gaseous Propane**

<i>T</i>	<i>k</i>	<i>T</i>	<i>k</i>
231.07	0.0114	400	0.0306
240	0.0121	420	0.0334
260	0.0139	440	0.0363
280	0.0159	460	0.0393
300	0.0180	480	0.0424
320	0.0202	500	0.0455
340	0.0226	520	0.0487
360	0.0252	540	0.0520
380	0.0278	560	0.0553

One of several models for this catalytic reaction, in which methane is adsorbed on the catalyst surface, is given by (model 1)

$$r_{\text{CO}_2} = k_s K_{\text{CH}_4} \left(P_{\text{CH}_4} P_{\text{H}_2\text{O}}^2 - \frac{P_{\text{CO}_2} P_{\text{H}_2}^4}{K_p} \right) \frac{1}{1 + K_{\text{CH}_4} P_{\text{CH}_4}}$$

Another model is a simply from of reversible reaction in which there is no component adsorption on the catalyst (model 2):

$$r_{\text{CO}_2} = k_l \left(P_{\text{CH}_4} P_{\text{H}_2\text{O}}^2 - \frac{P_{\text{CO}_2} P_{\text{H}_2}^4}{K_p} \right)$$

where $K_p = 5.051 \times 10^{-5}$ atm². Table P2.14 shows experimental results of reaction rate as a function of partial pressure of the products at 350°C.

TABLE P2.13
Heat of Vaporization of Propane

T	$10^{-7}\Delta H_v$	T	$10^{-7}\Delta H_v$
90	2.46	220	1.93
100	2.42	231.07	1.88
110	2.37	240	1.83
120	2.33	250	1.78
130	2.29	260	1.73
140	2.25	270	1.67
150	2.21	280	1.61
160	2.17	290	1.54
170	2.13	300	1.46
180	2.09	310	1.38
190	2.05	320	1.28
200	2.01	330	1.18
210	1.97	340	1.05

TABLE P2.14
Reaction Rate Data for Catalytic Reforming Reaction

n	$r_{CO_2} \times 10^3$ (gmol/hr/g)	Partial Pressure (atm)			
		CH ₄	H ₂ O	CO ₂	H ₂
1	0.13717	0.06298	0.23818	0.00420	0.01669
2	0.15584	0.03748	0.26315	0.00467	0.01686
3	0.20028	0.05178	0.29557	0.00542	0.02079
4	0.05700	0.04978	0.23239	0.00177	0.07865
5	0.20150	0.04809	0.29491	0.00655	0.02464
6	0.07887	0.03849	0.24171	0.00184	0.06873
7	0.14983	0.03886	0.26048	0.00381	0.01480
8	0.15988	0.05230	0.26286	0.05719	0.01635
9	0.26194	0.05185	0.33529	0.00718	0.02820
10	0.14426	0.06432	0.24787	0.00509	0.02055
11	0.20195	0.09609	0.28457	0.00652	0.02627

1. Estimate the values of parameters k_s , K_{CH_4} , and k_1 of models 1 and 2.
 2. Determine graphically which model best represents the given experimental data.
- 2.15** Suppose you have measured heights at a number of coordinates on the surface of a mountainous area. The area is 4 km wide (x) and 5 km long (y).

```

>> x = 0:0.5:4; % x(km)
>> y = 0:0.5:5; % y(km)
>> z = [95 93 96 94 97 94 93 94 96;
       96 94 93 94 97 93 96 94 94;
       93 94 92 91 96 93 97 96 98;
       95 93 92 91 93 95 96 96 93;
       97 96 93 92 95 98 99 97 98;
       97 98 97 96 98 99 98 96 95;
       93 97 95 96 98 102 101 100 98;

```

```

91 94 95 96 98 101 102 101 95;
95 97 98 96 97 98 97 95 94;
98 99 101 98 96 95 96 94 93;
94 95 96 96 95 94 93 94 95];

```

Estimate the height at the coordinate of $x = 1.7$ and $y = 3.6$.

- 2.16** Consider a long, rectangular flue with steady heat conduction along x- and y-axes. Figure P2.16 shows data of temperature distribution for the entire cross section of the rectangular flue with 11 ft wide (x) and 11 ft long (y).²² This data is stored in the spreadsheet Tdist.xlsx. Divide the ranges of x and y into 55 subintervals (i.e., the distance between adjacent nodes is 0.2 ft) and estimate the temperature at each node. Display the results graphically.
- 2.17** Solve the system of the Lorenz equations from $t = 0$ to 50:

$$\frac{d}{dt}y_1(t) = 10(y_2(t) - y_1(t))$$

$$\frac{d}{dt}y_2(t) = 28y_1(t) - y_2(t) - y_1(t)y_3(t)$$

$$\frac{d}{dt}y_3(t) = y_1(t)y_2(t) - \frac{8}{3}y_3(t)$$

The initial condition is $y(0) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$.

200	200	200	200	200	200	200	200	200	200	200	200	200
200	227.2	254.4	279.5	289.8	293	293	289.8	279.5	254.4	227.2	200	
200	254.4	312	369.6	387.9	392.2	392.2	387.9	369.6	312	254.4	200	
200	279.5	369.6	500	500	500	500	500	500	369.6	279.5	200	
200	289.8	387.9	500	500	500	500	500	500	387.9	289.8	200	
200	293	392.2	500	500	500	500	500	500	392.2	293	200	
200	293	392.2	500	500	500	500	500	500	392.2	293	200	
200	289.8	387.9	500	500	500	500	500	500	387.9	289.8	200	
200	279.5	369.6	500	500	500	500	500	500	369.6	279.5	200	
200	254.4	312	369.6	387.9	392.2	392.2	387.9	369.6	312	254.4	200	
200	227.2	254.4	279.5	289.8	293	293	289.8	279.5	254.4	227.2	200	
200	200	200	200	200	200	200	200	200	200	200	200	200

FIGURE P2.16 Temperature distribution for the cross section of a flue.

- 2.18** Figure P2.18 shows three continuous stirred heating tanks connected in series. Steam is used to heat the solution in each tank. The mass flow rate w_i ($i = 0, 1, 2, 3$) at each tank is maintained at a constant value of w kg/min. If the heat transfer area A is constant for each tank, the amount of heat transferred from the steam pipe to the solution is given by

$$Q_i = UA(T_s - T_i) \quad (i = 1, 2, 3)$$

The temperature at each tank before heating is T_0 . Using data given below, plot the temperature profile of each tank as a function of time. What is the steady-state temperature of the 3rd tank?

Data: Heat capacity of the solution $C_p = 1.75 \text{ kJ/(kg} \cdot ^\circ\text{C)}$; $UA = 12 \text{ kJ/(min} \cdot ^\circ\text{C)}$; $T_0 = 25^\circ\text{C}$; the mass of solution at each tank $M = 1400 \text{ kg}$; steam temperature $T_s = 200^\circ\text{C}$; the flow rate of the solution $w = 120 \text{ kg/min}$; time span $t = [0 \text{ } 200]$.

- 2.19** The system of the Robertson ordinary differential equations models a reaction between three chemical components²³:

$$\frac{dy_1(t)}{dt} = -0.04y_1(t) + 10^4 y_2(t)y_3(t)$$

$$\frac{dy_2(t)}{dt} = 0.04y_1(t) - 10^4 y_2(t)y_3(t) - 3 \times 10^7 y_2(t)^2$$

$$\frac{dy_3(t)}{dt} = 3 \times 10^7 y_2(t)^2$$

The Robertson reaction model is a typical example of the stiffness problem. Solve the Robertson ODE system from $t = 0$ to 3 and plot $y_2(t)$ versus time t . The initial conditions are $y_1(0) = 1$ and $y_2(0) = y_3(0) = 0$.

- 2.20** Consider a catalytic fluidized bed in which an irreversible gas phase reaction $A \rightarrow B$ occurs. The mass and energy balances along with the kinetic rate constant for this system are given as^{6,24}:

$$\frac{dP}{d\tau} = 0.1 + 320P_p - 321P, \quad \frac{dT}{d\tau} = 1752 - 269T + 267T_p,$$

$$\frac{dP_p}{d\tau} = 1.88 \times 10^3 \{P - P_p(1 + K)\}, \quad \frac{dT_p}{d\tau} = 1.3(T - T_p) + 1.04 \times 10^4 K P_p,$$

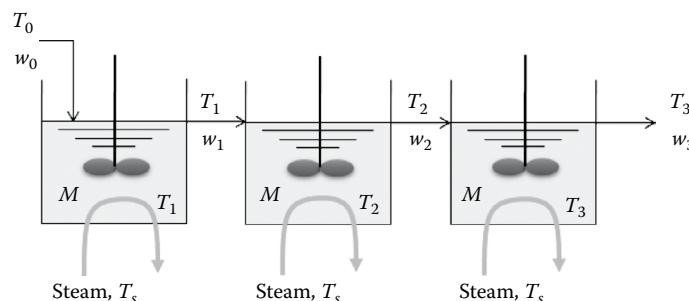


FIGURE P2.18 Continuous stirred heating tanks.

$$K = 6 \times 10^{-4} \exp\left(20.7 - \frac{1500}{T_p}\right)$$

where

T is the absolute temperature (°R)

P is the partial pressure of the reactant in the fluid (atm)

T_p is the temperature of the reactant at the catalyst surface (°R)

P_p is the partial pressure of the reactant at the catalyst surface (atm)

K is the dimensionless reaction rate constant

τ is the dimensionless time

Solve the equation system in the range of $0 \leq \tau \leq 1500$ and plot changes of dependent variables as functions of time. Use the initial values of $P(0) = 0.1$, $T(0) = 600$, $P_p = 0$, and $T_p = 761$.

- 2.21** Estimate the amount of heat (cal) required to heat 1 g mole of propane from 200°C to 700°C at 1 atm. The heat capacity of propane is given by

$$C_p = 2.41 + 0.057195T - 4.3 \times 10^{-6}T^2 \quad (T : \text{K}).d$$

REFERENCES

1. Kapuno, R. R. A., *Programming for Chemical Engineers*, Infinity Science Press, Hingham, MA, p. 122, 2008.
2. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 144, 2008.
3. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 686, 2008.
4. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 93, 2008.
5. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 203, 2008.
6. Aiken, R. C. and L. Lapidus, An effective integration method for typical stiff systems, *AICHE Journal*, 20(2), 368, 1974.
7. Shampine, L. F., I. Gladwell, and S. Thompson, *Solving ODEs with MATLAB*, Cambridge University Press, Cambridge, UK, pp. 168–169, 2003.
8. Skeel, R. D. and M. Berzins, A method for the spatial discretization of parabolic equations in one space variable, *SIAM Journal on Scientific and Statistical Computing*, 11, 1–32, 1990.
9. Martin, M. M., *Introduction to Software for Chemical Engineers*, CRC Press, Boca Raton, FL, p. 290, 2015.
10. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, p. 2, 1985.
11. Kano, M. and M. Ogawa, The state of the art in chemical process control in Japan: Good practice and questionnaire survey, *Journal of Process Control*, 11, 969–982, 2010.
12. Martin, M. M., *Introduction to Software for Chemical Engineers*, CRC Press, Boca Raton, FL, pp. 291–293, 2015.
13. Kapuno, R. R. A., *Programming for Chemical Engineers*, Infinity Science Press, Hingham, MA, p. 112, 2008.
14. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 23–24, 2008.
15. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 41, 2008.
16. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 685, 2008.
17. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 25, 2008.

18. Ingham, H., D. G. Friend, and J. F. Ely, Thermophysical properties of ethane, *Journal of Physical and Chemical Reference Data*, 20(2), 275–347, 1991.
19. Younglove, B. A. and J. F. Ely, Thermophysical properties of fluids. II. Methane, ethane, propane, isobutane and n-butane, *Journal of Physical and Chemical Reference Data*, 16, 577, 1987.
20. Haynes, W. M. and R. D. Goodwin, Thermophysical Properties of Propane from 85 to 700 K at Pressures to 70 MPa, National Bureau of Standards Monograph 170, Boulder, CO, 1982.
21. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 87–88, 2008.
22. Kapuno, R. R. A., *Programming for Chemical Engineers*, Infinity Science Press, Hingham, MA, pp. 240–241, 2008.
23. Higham, D. J. and N. J. Higham, *MATLAB Guide*, 2nd ed., SIAM, Philadelphia, PA, pp. 184–187, 2005.
24. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 207–208, 2008.

3 Physical Properties

Physical property data of chemical compounds are parameters frequently required for simulating chemical processes and in designing chemical plants. However, such data are not readily available outside of certain proprietary simulation design packages or large commercial databases. Physical property data can be obtained by conducting experiments to measure the properties of individual substances or of mixtures. However, because of the multitude of chemical compounds that are of interest to the chemical engineer and the vast number of thermodynamic states in which they appear in combination, it is impossible to obtain all physical property data by experiments alone. Therefore, it is necessary to establish generalizations for estimating physical properties of mixtures using only limited available experimental information.

The estimation of thermodynamic properties of vapors and liquids plays a very important role in chemical engineering computations. Numerous predictions and correlations of thermodynamic physical property data have been presented—and students, scientists, and practicing engineers alike have utilized these correlations and interpolations to obtain data of sufficient accuracy for their individual purposes.

In this chapter, physical property data and correlations for typical liquids and gases are introduced and reviewed. The correlations described in this chapter are limited to those that the author has deemed to have the greatest value in common practical application. The main objectives of this chapter are to provide readers with various estimation procedures and correlations for a limited set of properties of typical chemical compounds and to present MATLAB® programs that analyze properties for a range of variables and correlation constants.

3.1 WATER AND STEAM

Many studies have been conducted on the thermodynamic properties of water and steam. Various formulations that approximate properties of water and steam have been introduced through the years. In this section, we will focus on the computational procedures for properties of water and steam based on the formula depicted in the IAPWS-IF97 (International Association for Properties of Water and Steam-Industrial Formulation 1997)¹ revised in 2007. The range of validity covered in the IAPWS-IF97 is

$$273.15 \text{ K} \leq T \leq 1073.15 \text{ K}; P \leq 100 \text{ MPa}$$

$$1073.15 \text{ K} < T \leq 2273.15 \text{ K}; P \leq 50 \text{ MPa}$$

3.1.1 DIVISION OF PRESSURE-TEMPERATURE RANGE

The entire P - T range of validity of the IAPWS-IF97 is divided into five regions as shown in Figure 3.1. Each region is individually covered by fundamental equations for properties.

Region 1, which represents subcooled liquid under 623.15 K, and region 2, which represents superheated steam under 1073.15 K, are individually covered by a fundamental equation for the specific Gibbs free energy $g(P, T)$. Region 3 represents subcooled liquid and superheated steam located between regions 1 and 2 and is covered by a fundamental equation for the specific Helmholtz free energy $f(\rho, T)$. Region 4 is the saturated region and is covered by a fundamental equation in the form

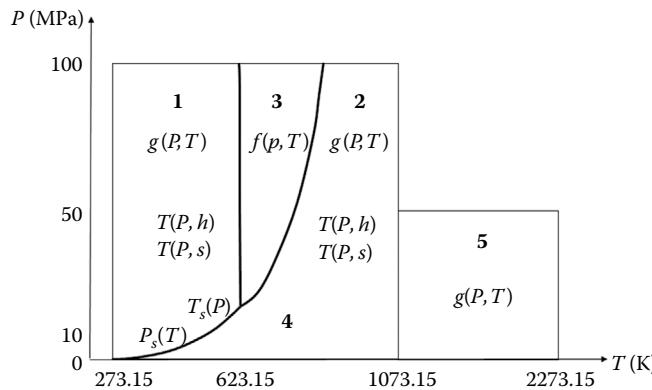


FIGURE 3.1 P - T diagram divided into five regions with property equations.

of $T_s(P)$. Region 5 represents superheated steam between 1073.15 and 2273.15 K and is covered by a fundamental equation for the specific Gibbs free energy $g(P, T)$. The maximum pressure of regions 1, 2, and 3 is 100 MPa, and that of region 5 is 50 MPa.

3.1.2 PROPERTY EQUATIONS

3.1.2.1 Parameters and Auxiliary Equations

The values of parameters and reference constants used in the IAPWS-IF97 are

Specific gas constant: $R = 0.461526 \text{ kJ/(kg} \cdot \text{K)}$

Critical temperature: $T_c = 647.096 \text{ K}$

Critical pressure: $P_c = 22.064 \text{ MPa}$

Critical density: $\rho_c = 322 \text{ kg/m}^3$

The specific internal energy and the specific entropy of the saturated liquid at the triple point are set equal to zero. This condition is met at the temperature and pressure of the triple point $T_t = 273.16 \text{ K}$, $P_t = 611.657 \text{ Pa}$ and at the enthalpy of the triple point $H_t = 0.611783 \text{ J/kg}$.

The boundary between regions 2 and 3 as shown in Figure 3.1 is defined by the following quadratic pressure-temperature relation:

$$\pi = n_1 + n_2\theta + n_3\theta^2$$

where $\pi = P/P^*$, $\theta = T/T^*$, $P^* = 1 \text{ MPa}$, $T^* = 1 \text{ K}$, and n_1 , n_2 , and n_3 are constants. This equation roughly describes an isentropic line. The IAPWS-IF97 involves many constants, and MATLAB programs presented in this book contain these constants.

3.1.2.2 Basic Equations for Region 1

The basic equation for region 1 is a fundamental equation for the specific Gibbs free energy g given by dimensionless form

$$\frac{g(P, T)}{RT} = \gamma(\pi, \tau) = \sum_{i=1}^{34} n_i (7.1 - \pi)^{I_i} (\tau - 1.222)^{J_i}$$

where $\pi = P/P^*$, $\tau = T^*/T$, $P^* = 16.53$ MPa, and n_i and J_i are constants. Other thermodynamic properties can be obtained from the above equation and its derivatives. For example, derivatives with respect to π and τ are used:

$$\left. \frac{\partial \gamma}{\partial \pi} \right|_{\tau} = - \sum_{i=1}^{34} n_i I_i (7.1 - \pi)^{I_i-1} (\tau - 1.222)^{J_i}, \quad \left. \frac{\partial \gamma}{\partial \tau} \right|_{\pi} = \sum_{i=1}^{34} n_i (7.1 - \pi)^{I_i} J_i (\tau - 1.222)^{J_i-1}$$

3.1.2.3 Basic Equations for Region 2

The basic equation for region 2 is a fundamental equation for the specific Gibbs free energy g . This equation can be expressed as a combination of an ideal gas part γ^o and a residual part γ^T :

$$\frac{g(P, T)}{RT} = \gamma(\pi, \tau) = \gamma^o(\pi, \tau) + \gamma^T(\pi, \tau)$$

$$\gamma^o = \ln \pi + \sum_{i=1}^9 n_i^o \tau^{J_i^o}, \quad \gamma^T = \sum_{i=1}^{43} n_i \pi^{I_i} (\tau - 0.5)^{J_i}, \quad \gamma_{\pi}^o = \frac{1}{\pi}, \quad \gamma_{\tau}^o = \sum_{i=1}^9 n_i^o J_i^o \tau^{J_i^o-1}$$

where $\pi = P/P^*$, $\tau = T^*/T$, $P^* = 1$ MPa, $T^* = 540$ K, and n_i , I_i , and J_i are constants. Derivatives of the ideal gas and residual parts with respect to π and τ are used in the calculation of physical properties:

$$\left. \frac{\partial \gamma^T}{\partial \pi} \right|_{\tau} = \sum_{i=1}^{43} n_i I_i \pi^{I_i-1} (\tau - 0.5)^{J_i}, \quad \left. \frac{\partial \gamma^T}{\partial \tau} \right|_{\pi} = \sum_{i=1}^{43} n_i \pi^{I_i} J_i (\tau - 0.5)^{J_i-1}$$

3.1.2.4 Basic Equations for Region 3

The basic equation for region 3 is a fundamental equation for the specific Helmholtz free energy f . This equation can be expressed in dimensionless form as

$$\frac{f(\rho, T)}{RT} = \Phi(\delta, \tau) = n_1 \ln \sigma + \sum_{i=2}^{40} n_i \delta^{I_i} \tau^{J_i}$$

where $\delta = \rho/\rho^*$, $\tau = T^*/T$, $\rho^* = \rho_c$, and $T^* = T_c$.

3.1.2.5 Basic Equations for Region 4

The saturation line can be expressed as a dimensionless quadratic equation, which can be solved in terms of saturation pressure P_s and saturation temperature T_s :

$$\beta^2 \vartheta^2 + n_1 \beta^2 \vartheta + n_2 \beta^2 + n_3 \beta \vartheta^2 + n_4 \beta \vartheta + n_5 \beta + n_6 \vartheta^2 + n_7 \vartheta + n_8 = 0$$

where $\beta = (P_s/P)^{1/4}$, $\vartheta = (T_s/T) + (n_9/((T_s/T^*) - n_{10}))$, $P^* = 1$ MPa, $T^* = 1$ K. The solution of the above equation for the saturation pressure yields

$$\frac{P_s}{P^*} = \left[\frac{2C}{-B + \sqrt{B^2 - 4AC}} \right]^4$$

where

$$A = \vartheta^2 + n_1 \vartheta + n_2, \quad B = n_3 \vartheta^2 + n_4 \vartheta + n_5, \quad C = n_6 \vartheta^2 + n_7 \vartheta + n_8$$

Similarly, the solution of the quadratic equation for the saturation temperature yields

$$\frac{T_s}{T^*} = \frac{n_{10} + D - \sqrt{(n_{10} + D)^2 - 4(n_9 + n_{10}D)}}{2}$$

where

$$D = \frac{2G}{-F - \sqrt{F^2 - 4EG}}, \quad E = \beta^2 + n_3\beta + n_6, \quad F = n_1\beta^2 + n_4\beta + n_7, \quad G = n_2\beta^2 + n_5\beta + n_8$$

3.1.2.6 Basic Equations for Region 5

The basic equation for high-temperature region 5 is a fundamental equation for the specific Gibbs free energy:

$$\frac{g(P, T)}{RT} = \gamma(\pi, \tau) = \gamma^o(\pi, \tau) + \gamma^T(\pi, \tau)$$

$$\gamma^o = \ln \pi + \sum_{i=1}^6 n_i^o \tau^{J_i^o}, \quad \gamma^T = \sum_{i=1}^6 n_i \pi^{I_i} \tau^{J_i}$$

where $\pi = P/P^*$, $\tau = T^*/T$, $P^* = 1$ MPa, $T^* = 1000$ K, and n_i , I_i , and J_i are constants.

The MATLAB function *H2OPT* is used to compute properties of water and steam. This function calculates thermodynamic properties of H₂O at the given temperature T (°C) and pressure P (kPa) for regions 1, 2, and 4. This function can be called as

```
wprop = H2OPT(P, T)
```

The returned parameter (wprop) is the name of the structure that stores the calculation results. wprop contains several fields: P is pressure (kPa), T is temperature (°C), v is specific volume (m³/kg), u is internal energy (kJ/kg), h is enthalpy (kJ/kg), and s is entropy (kJ/(kg · K)).

```
function [wprop] = H2OPT(P, T)
% Compute thermodynamic properties of H2O at the given temperature (C)
% and pressure(kPa) using the IAPWS formula(revised release on the
% IAPWS-97) (region 1, 2, 4)
% inputs
% P: kPa, T: C
% output:
% wprop: structure containing following fields:
% P(pressure, kPa), T(temperature, C), v(specific volume, m^3/kg),
% u(internal energy, kJ/kg), h(enthalpy, kJ/kg), s(entropy, kJ/kg/K)

% Initialization
crit = 1e-4;
Pg = 0; Tg = 0; vg = 0; ug = 0; sg = 0; hg = 0; % vapor phase
Pf = 0; Tf = 0; vf = 0; uf = 0; sf = 0; hf = 0; % liquid phase
stateH2O = ''; % state alarm message
warnmsg = ''; % alarm message
% Conversion of temperature and pressure units
T = T+273.15; % conversion of temperature unit: C -> K
```

```
P = P/1000; % conversion of pressure unit: kPa -> MPa
% identification of region(rg)
if T <= 623.15 % region 4
    Ts = rg4eqn(P);
    if (Ts-T) > crit
        rg = 1; % subcooled liquid
    elseif (T-Ts) > crit
        rg = 2; % superheated steam
    else
        rg = 3; % saturation state
    end
end
% auxiliary equation at the boundary between region 2 and 3
if T > 623.15 % boundary between region 2 and 3
    rg = 2; % saturated steam or outside of the region
    n1 = 348.05185628969; n2 = -1.1671859879975; n3 = 0.0010192970039326;
    Ps23 = n1 + n2*T + n3*T^2; % auxiliary equation at the boundary
    if P > Ps23
        rg = 5;
    end
end
if P < 1e-4 || T < 273.15
    rg = 4;
end
% region 1: subcooled liquid
if rg == 1 % T < Ts: subcooled liquid
    stateH2O = 'subcooled liquid';
    if T > 623.15 || P > 100
        warnmsg = 'Region not defined: P > 100 MPa'; rg = 6;
    else
        [Pf,Tf,vf,uf,sf,hf] = propPT(P,T,1); % region 1
    end
end
% region 2: superheated steam
if rg == 2 % T > Ts: superheated steam
    stateH2O = 'superheated steam';
    if T > 1073.15 || P > 100
        warnmsg = 'Region not defined: T > 800 C or P > 100 000 kPa';
        rg = 6;
    else
        [Pg,Tg,vg,ug,sg,hg] = propPT(P,T,2); % region 2
    end
end
% region 3: saturated vapor + saturated liquid
if rg == 3 % saturated state
    stateH2O = 'saturated state';
    warnmsg = '2 independent variables are required to determine the
    state';
    [Pg,Tg,vg,ug,sg,hg] = propPT(P,Ts,2); % vapor is in region 2
    [Pf,Tf,vf,uf,sf,hf] = propPT(P,Ts,1);
    % subcooled liquid is in region 1
end
% region 4
if rg == 4
    stateH2O = 'Incorrect data'; warnmsg = 'P or T < 0';
end
```

```

% region 5
if rg == 5 % outside region 3
    stateH2O = 'Calculation in region 3 by IAPWS-97 is not available';
    warnmsg = 'Check whether the pressure is greater than 100 MPa';
end
% restore units of temperature and pressure(K->C, MPa->kPa)
Tg = Tg - 273.15; Tf = Tf - 273.15; Pg = Pg*1000; Pf = Pf*1000;
% Assign values to corresponding fields of the structure to be returned.
if rg == 1 % region 1
    wprop.Pf = Pf; wprop.Tf = Tf; wprop.vf = vf;
    wprop.uf = uf; wprop.hf = hf; wprop.sf = sf;
elseif rg == 2 % region 2
    wprop.Pg = Pg; wprop.Tg = Tg; wprop.vg = vg;
    wprop.ug = ug; wprop.hg = hg; wprop.sg = sg;
elseif rg == 3 % region 3
    wprop.Pf = Pf; wprop.Tf = Tf; wprop.vf = vf;
    wprop.uf = uf; wprop.hf = hf; wprop.sf = sf;
    wprop.Pg = Pg; wprop.Tg = Tg; wprop.vg = vg;
    wprop.ug = ug; wprop.hg = hg; wprop.sg = sg;
end
wprop.stateH2O = stateH2O;
wprop.warnmsg = warnmsg;
% Print results.
fprintf('Temperature: %g C, Pressure: %g kPa\n', T-273.15, P*1000);
if rg == 1
    fprintf('liquid pressure = %g kPa\n', Pf); fprintf('liquid temp.
        = %g C\n', Tf);
    fprintf('volume = %g m^3/kg\n', vf);
    fprintf('internal energy = %g kJ/kg\n', uf);
    fprintf('enthalpy = %g kJ/kg\n', hf); fprintf('entropy = %g kJ/
        kg*K\n', sf);
    fprintf('state = %s\n', stateH2O);
elseif rg == 2
    fprintf('vapor pressure = %g kPa\n', Pg); fprintf('vapor temp.
        = %g C\n', Tg);
    fprintf('volume = %g m^3/kg\n', vg);
    fprintf('internal energy = %g kJ/kg\n', ug);
    fprintf('enthalpy = %g kJ/kg\n', hg); fprintf('entropy = %g kJ/
        kg*K\n', sg);
    fprintf('state = %s\n', stateH2O);
elseif rg == 3
    fprintf('liquid pressure = %g kPa\n', Pf); fprintf('liquid temp.
        = %g C\n', Tf);
    fprintf('vapor pressure = %g kPa\n', Pg); fprintf('vapor temp.
        = %g C\n', Tg);
    fprintf('liquid volume = %g m^3/kg\n', vf);
    fprintf('liquid internal energy = %g kJ/kg\n', uf);
    fprintf('liquid enthalpy = %g kJ/kg\n', hf);
    fprintf('liquid entropy = %g kJ/kg*K\n', sf);
    fprintf('vapor volume = %g m^3/kg\n', vg);
    fprintf('vapor internal energy = %g kJ/kg\n', ug);
    fprintf('vapor enthalpy = %g kJ/kg\n', hg);
    fprintf('vapor entropy = %g kJ/kg*K\n', sg);
    fprintf('state = %s\n', stateH2O);
end

```

Example 3.1: Calculation of H₂O Properties

Calculate the specific volume, the internal energy, the enthalpy, and the entropy of H₂O at the specified temperatures and pressures given below. Compare the results with those recorded in steam tables that can be found in standard thermodynamic texts.

1. 750 kPa, 167.76°C
2. 2100 kPa, 214.85°C

Solution

Results of calculations by the function *H2OPT* are shown below. These results match the values in steam tables:

```
>> wp = H2OPT(750,167.76);
Temperature: 167.76 C, Pressure: 750 kPa
vapor pressure = 750 kPa
vapor temp. = 167.76 C
volume = 0.255506 m^3/kg
internal energy = 2574.02 kJ/kg
enthalpy = 2765.65 kJ/kg
entropy = 6.68356 kJ/kg*K

>> wp = H2OPT(2100,214.85);
Temperature: 214.85 C, Pressure: 2100 kPa
liquid pressure = 2100 kPa
liquid temp. = 214.85 C
volume = 0.001181 m^3/kg
internal energy = 917.44 kJ/kg
enthalpy = 919.92 kJ/kg
entropy = 2.46999 kJ/kg*K
```

The function *H2OPT* calls another function *propPT*, which in turn calls other functions. Each function performs a specific task as shown in [Table 3.1](#). The functions shown in [Table 3.1](#) can be found in Appendix A.

TABLE 3.1
The MATLAB® Functions Used in the Calculations of H₂O Properties
at Given Temperature and Pressure

Name	Description
H2OPT	The main function: assigns the results to the structure variable
propPT	Calculates properties at regions 1, 2, and 4
rg1Jn	Defines 34 coefficients of the basic equation for region 1
rg2Jn	Defines coefficients (γ^o : 9, γ^T : 43) of the basic equation for region 2
rg1eqn	Calculates the basic equation for region 1
rg2eqn	Calculates the basic equation for region 2
rg4eqn	Calculates the basic equation for region 4
rg1gd _P	Calculates $(\partial\gamma/\partial\pi) _T$
rg1gd _T	Calculates $(\partial\gamma/\partial\tau) _\pi$

3.1.3 PROPERTIES OF SATURATED STEAM

In this section, we will focus on the thermodynamic properties of saturated steam valid from 273.15 to 647.15 K. The following description is based on the revised IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use.²

At a given temperature T (K), the saturated vapor pressure can be approximated as

$$p = p_c \exp \left[\frac{T_c}{T} \left(a_1 \tau + a_2 \tau^{1.5} + a_3 \tau^3 + a_4 \tau^{3.5} + a_5 \tau^4 + a_6 \tau^{7.5} \right) \right]$$

where $\tau = 1 - (T/T_c)$, p_c and T_c are critical pressure and critical temperature, respectively, and $a_i (i = 1, 2, \dots, 6)$ are constants. Differentiation of the vapor pressure with respect to the saturation temperature gives

$$\frac{dp}{dT} = \left(-\frac{p}{T} \right) \left[7.5a_6 \tau^{6.5} + 4a_5 \tau^3 + 3.5a_4 \tau^{2.5} + 3a_3 \tau^2 + 1.5a_2 \tau^{0.5} + a_1 + \ln \left(\frac{p}{p_c} \right) \right]$$

Alternatively, the saturation temperature can be calculated at a given pressure by converting the pressure equation into a nonlinear equation. By taking logarithm on the pressure equation, we have

$$\begin{aligned} \ln \left(\frac{p}{p_c} \right) &= \frac{T_c}{T} \left(a_1 \tau + a_2 \tau^{1.5} + a_3 \tau^3 + a_4 \tau^{3.5} a_5 \tau^4 + a_6 \tau^{7.5} \right) \\ &= \frac{1}{1 - \tau} \left(a_1 \tau + a_2 \tau^{1.5} + a_3 \tau^3 + a_4 \tau^{3.5} a_5 \tau^4 + a_6 \tau^{7.5} \right) \end{aligned}$$

Rearrangement of the above equation gives

$$f(\tau) = a_1 \tau + a_2 \tau^{1.5} + a_3 \tau^3 + a_4 \tau^{3.5} a_5 \tau^4 + a_6 \tau^{7.5} - (1 - \tau) \ln \left(\frac{p}{p_c} \right) = 0$$

At a given temperature T (K), the densities of the saturated liquid and steam can be represented respectively as

$$\rho_L = \rho_c \left[1 + b_1 \tau^{1/3} + b_2 \tau^{2/3} + b_3 \tau^{5/3} + b_4 \tau^{16/3} + b_5 \tau^{43/3} + b_6 \tau^{110/3} \right]$$

$$\rho_G = \rho_c \exp \left(c_1 \tau^{1/3} + c_2 \tau^{2/3} + c_3 \tau^{4/3} + c_4 \tau^3 + c_5 \tau^{37/6} + c_6 \tau^{71/6} \right)$$

where $c_i (i = 1, 2, \dots, 6)$ are constants.

The specific enthalpy of the saturated liquid and steam at a given temperature T (K) can be obtained, respectively, from the following relations:

$$h_L = \alpha + \frac{T}{\rho_L} \frac{dp}{dT}, \quad h_G = \alpha + \frac{T}{\rho_G} \frac{dp}{dT}$$

$$\alpha = \alpha_0 \left[d_a + d_1 \theta^{-19} + d_2 \theta + d_3 \theta^{4.5} + d_4 \theta^5 + d_5 \theta^{54.5} \right]$$

where $d_i (i = 1, 2, \dots, 5)$ are constants.

The specific entropy of the saturated liquid and steam at a given temperature T (K) can be obtained, respectively, from the following equations:

$$s_L = \phi + \frac{1}{\rho_L} \frac{dp}{dT}, \quad s_G = \phi + \frac{1}{\rho_G} \frac{dp}{dT}$$

$$\phi = \phi_0 \left[d_\phi + \frac{19}{20} d_1 \theta^{-20} + d_2 \ln \theta + \frac{9}{7} d_3 \theta^{3.5} + \frac{5}{4} d_4 \theta^4 + \frac{109}{107} d_5 \theta^{53.5} \right]$$

The MATLAB function *satsteam* calculates the properties of the saturated steam. This function calculates the properties of the saturated steam at the given temperature T_s (°C), and the results produced by this function are stored in the structure *stpr*. The calling format of this function is

```
stpr = satsteam(Ts)
```

stpr contains several fields: *stpr.T* = temperature of the saturated steam (K); *stpr.P* = pressure (Pa); *stpr.vL* = liquid volume (m³/kg); *stpr.vV* = steam volume (m³/kg); *stpr.hL* = liquid enthalpy (J/kg); *stpr.hV* = steam enthalpy (J/kg); *stpr.sL* = liquid entropy (J/(kg·K)); *stpr.sV* = steam entropy (J/(kg·K)).

```
function stpr = satsteam(Ts)
% Computes properties of the saturated steam at a given temperature Ts (C)
% stpr: a structure containing results of calculations

% Critical properties
Tc = 647.096; % critical temperature(K)
Pc = 22064000; % critical pressure(Pa)
rhoc = 322; % critical density(kg/m^3)

% Definition of parameters and constants
Ts = Ts + 273.15;
alpha0 = 1000; % alpha_0 (J/kg)
phi0 = 1000/647.096;
a = [-7.85951783 1.84408259 -11.7866497 22.6807411...
       -15.9618719 1.80122502];
b = [1.99274064 1.09965342 -0.510839303 -1.75493479...
       -45.5170352 -674694.45];
c = [-2.03150240 -2.68302940 -5.38626492 -17.2991605...
       -44.7586581 -63.9201063];
d = [-5.65134998e-8 2690.66631 127.287297 -135.003439 0.981825814];
alphad = -1135.905627715;
phid = 2319.5246;
theta = Ts/Tc;
tau = 1 - theta;

% saturated steam pressure
tw = (Tc/Ts)*(a(1)*tau + a(2)*tau^1.5 + a(3)*tau^3+...
               a(4)*tau^3.5 + a(5)*tau^4 + a(6)*tau^7.5);
Ps = Pc*exp(tw);

% density of saturated liquid
rhoL =rhoc*(1 + b(1)*tau^(1/3) + b(2)*tau^(2/3) + b(3)*tau^(5/3) +...
               b(4)*tau^(16/3) + b(5)*tau^(43/3) + b(6)*tau^(110/3));

% density of saturated steam
vw = c(1)*tau^(1/3) + c(2)*tau^(2/3) + c(3)*tau^(4/3) +...
```

```

c(4)*tau^3 + c(5)*tau^(37/6) + c(6)*tau^(71/6);
rhoV = rhoc*exp(vw);

% specific volume
vL = 1/rhoL; % saturated liquid
vV = 1/rhoV; % saturated steam

% alpha
alpha = alpha0*(alphad + d(1)*theta^(-19) + d(2)*theta +...
    d(3)*theta^4.5 + d(4)*theta^5 + d(5)*theta^54.5);

% phi
phi = phi0*(phid + (19/20)*d(1)*theta^(-20) + d(2)*log(theta) +...
    (9/7)*d(3)*theta^3.5 + (5/4)*d(4)*theta^4 +...
    (109/117)*d(5)*theta^53.5);

% dp/dT
tv = 7.5*a(6)*tau^6.5 + 4*a(5)*tau^3 + 3.5*a(4)*tau^2.5 +...
    3*a(3)*tau^2 + 1.5*a(2)*tau^0.5 + a(1) + log(Ps/Pc);
dpdT = (-Ps/Ts)*tv;

% enthalpy
hL = alpha + (Ts/rhoL)*dpdT; % saturated liquid
hV = alpha + (Ts/rhoV)*dpdT; % saturated steam

% entropy
sL = phi + (1/rhoL)*dpdT; % saturated liquid
sV = phi + (1/rhoV)*dpdT; % saturated steam

% result structure
stpr.T = Ts;
stpr.P = Ps;
stpr.vL = vL;
stpr.vV = vV;
stpr.hL = hL;
stpr.hV = hV;
stpr.sL = sL;
stpr.sV = sV;
end

```

Example 3.2: Physical Properties of the Saturated Steam

Calculate the specific volume, the enthalpy, and the entropy of the saturated steam at 150°C. Compare the results with those recorded in steam tables that be found in standard thermodynamic texts.

Solution

Next is an example of calling the function *satsteam* and storing the results in a structure called *stpr*. Outputs of this function are displayed by using the built-in function *fprintf* and are summarized in [Table 3.2](#).

```

>> stpr = satsteam(150);
>> fprintf('Temperature(saturated steam): %g \n', stpr.T-273.15);
>> fprintf('Pressure = %g kPa\n', stpr.P/1000);
>> fprintf('Liquid volume = %g cm^3/g\n', stpr.vL*1000);
>> fprintf('Steam volume = %g cm^3/g\n', stpr.vV*1000);
>> fprintf('Liquid enthalpy = %g kJ/kg\n', stpr.hL/1000);

```

TABLE 3.2
Results of Physical Property Calculations of the Saturated Steam

	Pressure (kPa)	Liquid Volume (cm ³ /g)	Steam Volume (cm ³ /g)	Liquid Enthalpy (kJ/kg)	Steam Enthalpy (kJ/kg)	Liquid Entropy (kJ/(kg · K))	Steam Entropy (kJ/(kg · K))
Equations	476.159	1.0905	392.48	632.147	2746.05	1.84172	6.83735
Steam table ³	476	1.091	392.4	632.1	2745.4	1.8416	6.8358

```
>> fprintf('Steam enthalpy = %g kJ/kg\n', stpr.hV/1000);
>> fprintf('Liquid entropy = %g kJ/kg*K\n', stpr.sL/1000);
>> fprintf('Steam entropy = %g kJ/kg*K\n', stpr.sV/1000);
```

```
Temperature(saturated steam) : 150 C
Pressure = 476.159 kPa
Liquid volume = 1.0905 cm^3/g
Steam volume = 392.48 cm^3/g
Liquid enthalpy = 632.147 kJ/kg
Steam enthalpy = 2746.05 kJ/kg
Liquid entropy = 1.84172 kJ/kg*K
Steam entropy = 6.83735 kJ/kg*K
```

3.2 HUMIDITY

3.2.1 RELATIVE HUMIDITY

Relative humidity (H_R) is defined as the ratio of the pressure exerted by the water vapor to the vapor pressure of the liquid water taken at that temperature. In general, relative humidity is represented in terms of percentage (%).

$$H_R = \frac{100p_v}{P_w} (\%)$$

where

p_v is the partial pressure of water vapor

P_w is the vapor pressure of water at the given temperature

This relation can also be expressed as

$$H_R = \frac{100P_a}{P_s}$$

where

P_a is the actual vapor pressure

P_s is the saturation vapor pressure at the given temperature

At a given temperature T (K), the saturated vapor pressure P_s can be approximated as

$$P_s = P_c \exp \left[\frac{T_c}{T} \left(a_1 \tau + a_2 \tau^{1.5} + a_3 \tau^3 + a_4 \tau^{3.5} + a_5 \tau^4 + a_6 \tau^{7.5} \right) \right]$$

where $\tau = 1 - (T/T_c)$, P_c and T_c are critical pressure and critical temperature, respectively, and $a_i (i = 1, 2, \dots, 6)$ are constants.

3.2.2 ABSOLUTE HUMIDITY

Absolute humidity (H_A) is defined as the mass of water vapor per unit mass of vapor-free air. When the total pressure (sum of the partial pressure of vapor and air) is constant, absolute humidity depends solely on the pressure exerted by the water vapor in the vapor-air mixture.

$$H_A = \frac{M_w p_a}{M_a p_w} = \frac{M_w p_a}{M_a (P_T - p_a)}$$

where

M_a and M_w are mol fractions of air and water vapor, respectively

p_a and p_w are partial pressures of air and water vapor, respectively

P_T is the total pressure and can be taken as atmospheric pressure

H_A can be estimated by using the ideal gas equation of state. Since H_A can be represented as the density of water vapor in the air, we have from the ideal gas equation

$$H_A = \frac{p_a}{R_w T}$$

where $R_w = 461.5 \text{ J/(kg}\cdot\text{K)}$.

Calculation of absolute humidity is performed by the MATLAB function *humidest*. This function computes absolute humidity and the vapor pressure using relative humidity $H_R(\%)$ and the dry bulb temperature $T_d(\text{°C})$. The basic function call is

```
hum = humidest(Hr, Td)
```

where H_r is relative humidity (%) and T_d is the dry bulb temperature (°C). This function returns a structure that contains actual vapor pressure (hum.pres, Pa) and absolute humidity (hum.Ha, g/m³).

```
function hum = humidest(Hr, Td)
% Hr: relative humidity
% Td: dry bulb temperature

% critical properties
Tc = 647.096; % critical temperature (K)
Pc = 22064000; % critical pressure (Pa)
% constants
Rw = 461.512244565;
a = [-7.85951783 1.84408259 -11.7866497 22.6807411...
-15.9618719 1.80122502];
% temperature parameters
Td = Td+273.15; theta = Td/Tc; tau = 1 - theta;
% saturated pressure
tw = (Tc/Td)*(a(1)*tau + a(2)*tau^1.5 + a(3)*tau^3+...
a(4)*tau^3.5 + a(5)*tau^4 + a(6)*tau^7.5);
Ps = Pc*exp(tw);
% results
hum.pres = (Hr/100)*Ps; % actual vapor pressure
hum.Ha = hum.pres*1000/((Td)*Rw); % absolute humidity
fprintf('Actual vapor pressure: %g Pa\n', hum.pres);
fprintf('Absolute humidity = %g g/m^3\n', hum.Ha);
end
```

Example 3.3: Humidity of Saturated Water Vapor

Calculate the actual vapor pressure (Pa) and absolute humidity (g/m^3) when the dry bulb temperature is 40°C and relative humidity is 50%.

Solution

Call to the function `humidest` with $\text{Hr} = 50$ and $\text{Td} = 40$ follows:

```
>> hum = humidest(50,40);
Actual vapor pressure: 3692.56 Pa
Absolute humidity = 25.55 g/m^3
```

3.3 DENSITY OF SATURATED LIQUIDS

Densities of saturated liquids at any given temperature can be given by the correlation⁴

$$\rho_L = AB^{-(1-T_r)^{2/7}}$$

where

ρ_L is the saturated liquid density (g/cm^3)

A and B are the correlation constants for a chemical compound being considered

T_c is the critical temperature ($^\circ\text{C}$)

$T_r = (T/T_c)$ is reduced temperature

Table 3.3 lists typical components whose constants A and B as well as critical temperatures are known. Each component in **Table 3.3** is assigned a specific identification number by the MATLAB function `compID`.

```
function ind = compID(cname)
% Assignment of identification number
cname = upper(cname);
switch cname
    case {'FLUORINE','F2'}, ind = 1;
    case {'CHLORINE','Cl2'}, ind = 2;
    case {'SULFUR DIOXIDE','SO2'}, ind = 3;
```

TABLE 3.3
Typical Chemical Compounds with Known Constants A and B

1	Fluorine (F_2)	10	Hydrogen (H_2)	19	Aniline ($\text{C}_6\text{H}_7\text{N}$)
2	Chlorine (Cl_2)	11	Nitrogen (N_2)	20	Phenol ($\text{C}_6\text{H}_5\text{O}$)
3	Sulfur dioxide (SO_2)	12	Oxygen (O_2)	21	Cyclopropane (C_3H_6)
4	Carbon monoxide (CO)	13	Ethylene (C_2H_4)	22	Cyclohexane (C_6H_{12})
5	Carbon dioxide (CO_2)	14	Methane (CH_4)	23	1,3 Butadiene (C_4H_6)
6	Hydrogen chloride (HCl)	15	Ethane (C_2H_6)	24	Methanol (CH_3O)
7	Ammonia (NH_3)	16	Propane (C_3H_8)	25	Chloroform (CHCl_3)
8	Water (H_2O)	17	Benzene (C_6H_6)	26	Carbon tetrachloride (CCl_4)
9	Hydrogen peroxide (H_2O_2)	18	Toluene (C_7H_8)		

Source: Yaws, C.L. et al., *Physical Properties*, A Chemical Engineering Publication, McGraw-Hill, New York, 1977.

```

case {'CARBON MONOXIDE','CO'}, ind = 4;
case {'CARBON DIOXIDE','CO2'}, ind = 5;
case {'HYDROGEN CHLORIDE','HCl'}, ind = 6;
case {'AMMONIA','NH3'}, ind = 7;
case {'WATER','H2O'}, ind = 8;
case {'HYDROGEN PEROXIDE','H2O2'}, ind = 9;
case {'HYDROGEN','H2'}, ind = 10;
case {'NITROGEN','N2'}, ind = 11;
case {'OXYGEN','O2'}, ind = 12;
case {'ETHYLENE','C2H4'}, ind = 13;
case {'METHANE','CH4'}, ind = 14;
case {'ETHANE','C2H6'}, ind = 15;
case {'PROPANE','C3H8'}, ind = 16;
case {'BENZENE','C6H6'}, ind = 17;
case {'TOLUENE','C7H8'}, ind = 18;
case {'ANILINE','C6H7N'}, ind = 19;
case {'PHENOL','C6H6O'}, ind = 20;
case {'CYCLOPROPANE','C3H6'}, ind = 21;
case {'CYCLOHEXANE','C6H12'}, ind = 22;
case {'1,3 BUTADIENE','C4H6'}, ind = 23;
case {'METHANOL','CH4O'}, ind = 24;
case {'CHLOROFORM','CHCl3'}, ind = 25;
case {'CARBON TETRACHLORIDE','CCl4'}, ind = 26;
end

```

The MATLAB function *denL* calculates the density of saturated liquids at a given temperature. This function specifies constants *A* and *B* and critical temperatures for the compounds listed in [Table 3.3](#). This function has the general syntax

```
rhoL = denL(T, cname)
```

where *T* is a given temperature (a scalar) or temperature range (a row vector) and *cname* is the name of the component that can be specified either as chemical formulas (such as 'C6H6O') or common names (such as "Phenol"). The parameters of this function are not case sensitive. This function calls the function *compID* to assign a unique identification number to the compound *cname*.

```

function rhoL = denL(T,cname)
% density of saturated liquid
% input
%   T: temperature(C) (scalar or row vector)
%   Tc: critical temperature(C)
%   cname: name or chemical formula of the compound
% output
%   rhoL: density of saturated liquid(g/cm^3)

% coefficients A and B of the density correlation equation
coefAB = [0.5649    0.2828    -129.0000
           0.5615    0.2720    144.0000
           0.5164    0.2554    157.6000
           0.2931    0.2706    -140.1000
           0.4576    0.2590    31.1000
           0.4183    0.2619    51.5000
           0.2312    0.2471    132.4000

```

```

0.3471    0.2740    374.2000
0.0        0.0        0.0 % missing component (H2O2)
0.0315    0.3473    -240.2000
0.3026    0.2763    -146.8000
0.4227    0.2797    -118.5000
0.2118    0.2784    9.9000
0.1611    0.2877    -82.6000
0.2202    0.3041    32.3000
0.2204    0.2753    96.7000
0.3051    0.2714    288.9400
0.2883    0.2624    318.8000
0.3392    0.2761    426.0000
0.4094    0.3246    420.0000
0.2614    0.2826    124.9000
0.2729    0.2727    280.3000
0.2444    0.2710    152.0000
0.2928    0.2760    239.4000
0.5165    0.2666    263.4000
0.5591    0.2736    283.2000];
ind = compID(cname); % identification number of the component
% compute density
Tr = T./coefAB(ind,3); epn = -(1 - Tr).^(2/7);
rhoL = coefAB(ind,1) * coefAB(ind,2).^epn;
end

```

Example 3.4: Density of Water

Calculate the density of saturated water at 150°C.

Solution

```

>> rw = denL(150, 'Water')
rw =
1.0621

```

3.3.1 COSTALD METHOD

For the estimating liquid densities, the COSTALD method can be used.⁵ This method can be applied for T_r within the range $0.25 < T_r < 1$.

$$v_s = \frac{1}{\rho} = v_c \cdot V_R^{(0)} \left(1 - w V_R^{(\delta)} \right)$$

$$V_R^{(0)} = 1 + a(1 - T_r)^{1/3} + b(1 - T_r)^{2/3} + c(1 - T_r) + d(1 - T_r)^{4/3}$$

$$V_R^{(\delta)} = \frac{e + fT_r + gT_r^2 + T_r^3}{T_r - 1.00001}$$

where $a = -1.52816$, $b = 1.43907$, $c = -0.81446$, $d = 0.190454$, $e = -0.296123$, $f = 0.386914$, $g = -0.0427458$, and v_s is the specific volume (cm^3/mol) and v_c is the critical

volume (cm³/mol). The MATLAB function *costald* performs the COSTALD method. The basic calling syntax is

```
rhoL = costald(T, Tc, vc, w, Mw)
```

where T is the given temperature, Tc is the critical temperature, vc is the critical volume, w is the acentric factor, Mw is the molecular weight (g/mol), and rhoL is the estimated density (kg/m³).

```
function rhoL = costald(T, Tc, vc, w, Mw)
% estimation of liquid density by COSTALD method
% input:
%   T, Tc: temperature and critical temperature (K)
%   vc: critical volume
%   w: acentric factor
%   Mw: molecular weight (g/mol)
% output
%   rhoL: estimated density (kg/m^3)

a = -1.52816; b = 1.43907; c = -0.81446; d = 0.190454;
e = -0.296123; f = 0.386914; g = -0.0427458; h = -0.0480645;
Tr = T./Tc;
Vr0 = 1+a.* (1-Tr).^(1/3)+b.* (1-Tr).^(2/3)+c.* (1-Tr)+d.* (1-Tr).^(4/3);
Vrd = (e + f*Tr + g*Tr.^2 + h*Tr.^3)./(Tr - 1.00001);
spV = vc.*Vr0.* (1 - w.*Vrd); % specific volume(cm^3/mol)
rhoL = 1000*Mw/spV; % density(kg/m^3)
end
```

Example 3.5: Density of *n*-Hexane⁶

Estimate the liquid density of *n*-hexane at $T = 293.15$ K using the COSTALD method. For *n*-hexane, $T_c = 507.8$ K, $v_c = 386.8$ cm³/mol, $w = 0.3002$, and $M_w = 86.178$ g/mol.

Solution

```
>> T=293.15; Tc=507.8; vc=386.8; w=0.3002; Mw = 86.178;
>> rhoL = costald(T, Tc, vc, w, Mw)
rhoL =
629.6425
```

The experimental value is 659.4 kg/m³.

3.4 VISCOSITY

3.4.1 VISCOSITY OF LIQUIDS

The viscosity of a saturated liquid can be expressed as a function of temperature:

$$\log \mu_L = A + \frac{B}{T} + CT + DT^2$$

where

μ_L is the viscosity of a saturated liquid (cP)

A, B, C, and D are correlation constants

T is the temperature (K)

Viscosities of liquids decrease with temperature, and this variation is almost linear over a wide range of temperatures from the freezing point to the boiling point. This phenomenon can be represented by the Andrade correlation⁷:

$$\ln \mu_L = \ln A + \frac{B}{T}$$

The MATLAB function *visL* calculates the viscosity of liquids. This function specifies constants *A*, *B*, *C*, and *D* for typical compounds listed in [Table 3.3](#). This function can be used as follows:

```
mu = visL(T, cname)
```

where *T* is the temperature (K); *cname* is the name of the component, which can be specified as a chemical formula or common name; and *mu* is the resultant viscosity of the saturated liquid (cP). *T* can be a scalar or a row vector representing a temperature range. This function calls the function *compID* to assign an identification number to the compound *cname*.

```
function mu = visL(T,cname)
% Calculates viscosity of a saturated liquid (cP)
% input
%   T: temperature(K) (scalar or a row vector)
%   cname: common name or chemical formula of the compound
% output
%   mu: the viscosity of the saturated liquid (cP)

% correlation constants (A, B, C, D)
wv = 1.0e+03 * [-0.001576  0.085630  -0.0000004073  -0.000000002725
                 -0.0007681  0.151400  -0.00000080650  0.000000000407500
                 -0.0026700  0.406700  0.00000614100  -0.000000012540000
                 -0.0023460  0.105200  0.00000461300  -0.000000019640000
                 -0.0013450  0.021220  0.00001034000  -0.000000034050000
                 -0.0015150  0.194600  0.00000306700  -0.000000013760000
                 -0.0085910  0.876400  0.00002681000  -0.000000036120000
                 -0.0107300  1.82800  0.00001966000  -0.000000014660000
                 0.0        0.0        0.0        0.0        % missing component (H2O2)
                 -0.0048570  0.025130  0.00014090000  -0.000002773000000
                 -0.0121400  0.376100  0.00012000000  -0.000000470900000
                 -0.0020720  0.093220  0.00000603100  -0.000000027210000
                 -0.0077060  0.468100  0.00003725000  -0.000000076330000
                 -0.0116700  0.499300  0.00008125000  -0.000000022630000
                 -0.0044440  0.290100  0.00001905000  -0.000000041640000
                 -0.0033720  0.313500  0.00001034000  -0.000000020260000
                 0.0020030  0.064660  -0.00001105000  0.000000009648000
                 0.0        0.0        0.0        0.0        % missing component (C6H6)
                 0.0        0.0        0.0        0.0        % missing component (C6H6O)
                 -0.0080390  1.889000  0.00001055000  -0.000000006718000
                 -0.0013350  0.1162000 0.0000001108  -0.000000000038360
                 -0.0019100  0.5992000  -0.0000006749  0.000000000502600
                 -0.0026370  0.4345000  0.0000019370  -0.000000002907000
                 -0.0026970  0.7009000  0.0000026820  -0.000000004917000
                 -0.0018120  0.3975000  0.0000011740  -0.000000001784000
                 -0.0056580  0.9945000  0.0000101600  -0.000000008733000];
ind = compID(cname); % identification number of the component
% compute viscosity
pn = wv(ind,1) + wv(ind,2)./T + wv(ind,3).*T + wv(ind,4).*T.^2;
mu = 10.^pn;
end
```

3.4.2 VISCOSITY OF GASES

The viscosity of gases at low pressures can be expressed by⁸

$$\mu_G = 0.0001^* (A + BT + CT^2)$$

where

μ_G is the viscosity of the gas at low pressure (cP)

T is the temperature (K)

A , B , and C are correlation constants for a chemical compound

Viscosities of gas mixtures at low pressures can be estimated by the method of Wilke,⁹ with Herning–Zipperer approximation¹⁰:

$$\mu_m = \frac{\sum y_i \mu_i (MW_i)^{0.5}}{\sum y_i (MW_i)^{0.5}}$$

where

μ_m is the gas mixture viscosity (cP)

y_i is the mole fraction of component i

μ_i is the gas viscosity of component i

MW_i is the molecular weight of component i

The MATLAB function *visG* calculates the viscosity of gases listed in Table 3.3. This function specifies constants A , B , and C for typical compounds listed in Table 3.3. The basic calling syntax is

```
mu = visG(T, cname)
```

where T is the temperature (K), $cname$ is the name of the component that can be specified as a chemical formula or common name, and mu is the resultant viscosity of the gas (cP). T can be a scalar or a row vector representing a temperature range. This function calls the function *compID* to assign an identification number to the compound $cname$.

```
function mu = visG(T,cname)
% Calculates viscosity of a gas (cP)
% input
%   T: temperature(K) (scalar or a row vector)
%   cname: common name or chemical formula of the compound
% output
%   mu: the viscosity of the gas (cP)

% correlation constants (A, B, C)
cv = [22.0900 76.9000 -211.6000
      5.1750 45.6900 -88.5400
      -3.7930 46.4500 -72.7600
      32.2800 47.4700 -96.4800
      25.4500 45.4900 -86.4900
      -9.5540 54.4500 -96.5600
      -9.3720 38.9900 -44.0500
      -31.8900 41.4500 -8.2720
      5.3810 28.9800 38.4000
      21.8700 22.2000 -37.5100
      30.4300 49.8900 -109.3000]
```

```

18.1100  66.3200 -187.9000
 3.5860  35.1300 -80.5500
15.9600  34.3900 -81.4000
 5.5760  30.6400 -53.0700
 4.9120  27.1200 -38.0600
-15.7600  32.4500 -72.3200
-8.4210  27.1100 -40.1800
-14.9800  29.0300 -1.1160
-16.4100  32.0000      0
-7.7870  34.7800 -81.3000
-4.7050  26.3200 -44.1000
-10.6700  34.3200 -80.8000
-5.6360  34.4500 -3.3400
-6.6880  37.2600 -50.8700
 5.6980  32.7300 -40.2800];
wv = [cv(:,1) cv(:,2)*1e-2 cv(:,3)*1e-6];
ind = compID(cname); % identification number of the component
% compute viscosity
pn = wv(ind,1) + wv(ind,2).*T + wv(ind,3).*T.^2;
mu = pn*1e-4;
end

```

Example 3.6: Viscosity of Phenol

Estimate the viscosity of liquid phenol at 150°C.

Solution

```

>> T = 150+273.15;
>> mu = visL(T, 'C6H6O')
mu =
 0.4858

```

Example 3.7: Viscosity of Methane

Estimate the viscosity of methane gas at 100°C.

Solution

```

>> T = 100+273.15; cname = 'methane';
>> mu = visG(T, cname)
mu =
 0.0133

```

3.5 HEAT CAPACITY

3.5.1 HEAT CAPACITY OF LIQUIDS

3.5.1.1 Polynomial Correlation

Liquid heat capacity can be expressed as polynomial of the form

$$C_p^L = A + BT + CT^2 + DT^3$$

where

C_p^L is the heat capacity of saturated liquid (cal/(g·°C))

T is the temperature (K)

A, B, C , and D are correlation constants for a chemical compound

Liquid heat capacities are not strongly dependent upon temperatures except in the range of $0.7 \leq T_r \leq 0.8$, where T_r is the reduced temperature.¹¹ But, at high T_r , liquid heat capacities become large and strongly dependent upon temperatures. At the boiling point of most organic compounds, heat capacities are between 0.4 and 0.5 cal/(g·K).

The MATLAB function *hcapL* calculates liquid heat capacities for the compounds listed in [Table 3.3](#). This function specifies constants *A*, *B*, *C*, and *D* for typical compounds listed in [Table 3.3](#). This function can be used as

```
hcp = hcapL(T, cname)
```

where *T* is the temperature (K); *cname* is the name of the component, which can be specified as a chemical formula or common name; and *hcp* is the resultant heat capacity of the liquid component (cal/(g·°C)). *T* can be either a scalar or a row vector representing a temperature range. This function calls the function *compID* to assign an identification number to the compound *cname*.

```
function hcp = hcapL(T, cname)
% Calculates liquid heat capacity (cal/g/C)
% input
%   T: temperature(K) (scalar or a row vector)
%   cname: common name or chemical formula of the compound
% output
%   hcp: liquid heat capacity(cal/g/C)

% correlation constants (A, B, C, D)
cv = 1.0e+04 * [-0.0000288  0.02528  -0.0331  0.1464
                 -0.000013220  0.0004720  -0.0020370  0.002894000
                 -0.000057370  0.0010340  -0.0040280  0.005285000
                 0.000056450  0.0004798  -0.0143700  0.091195000
                 -0.001930000  0.0254600  -0.1095500  0.157330000
                 -0.000011210  0.0007048  -0.0035310  0.006621000
                 -0.000192300  0.0031100  -0.0110900  0.013760000
                 0.000067410  0.0002825  -0.0008371  0.000860100
                 0.000044400  0.0001199  -0.0002738  0.000261500
                 0.000379000  -0.0329800  1.2170900  -0.243480000
                 -0.000106400  0.0059470  -0.0768700  0.335730000
                 -0.000045870  0.0032340  -0.0395100  0.157570000
                 -0.000034020  0.0006218  -0.0050120  0.012630000
                 0.000123000  -0.0010330  0.0072000  -0.010730000
                 0.000013880  0.0008481  -0.0056540  0.012610000
                 0.000033260  0.0002332  -0.0013360  0.003016000
                 -0.000148100  0.0015460  -0.0043700  0.004409000
                 -0.000014610  0.0004584  -0.0013460  0.001425000
                 0.000014070  0.0002467  -0.0006085  0.000592700
                 -0.000068960  0.0008218  -0.0018420  0.001447000
                 -0.000002618  0.0006913  -0.0034770  0.005990000
                 -0.000128400  0.0013390  -0.0035100  0.003227000
                 0.000037850  0.0001049  -0.0005761  0.001374000
                 0.000083820  -0.0003231  0.0008296  -0.000016890
                 -0.000009154  0.0003149  -0.0010640  0.001240000
                 -0.000001228  0.0002058  -0.0007040  0.000861000];
wv = [cv(:,1) cv(:,2)*1e-3 cv(:,3)*1e-6 cv(:,4)*1e-9];
ind = compID(cname); % identification number of the component
% computes liquid heat capacity
hcp = wv(ind,1) + wv(ind,2).*T + wv(ind,3).*T.^2 + wv(ind,4).*T.^3;
end
```

3.5.1.2 Rowlinson/Bondi Method

The method of Rowlinson/Bondi can be used to estimate liquid heat capacities. This method is based on the ideal gas heat capacity and the corresponding states principle¹²:

$$C_p^L = C_p^{id} + 1.45R + \frac{0.45R}{1-T_r} + 0.25wR \left(17.11 + \frac{25.2(1-T_r)^{1/3}}{T_r} + \frac{1.742}{1-T_r} \right)$$

where C_p^{id} is the ideal gas heat capacity. The MATLAB function *hcRB* performs the calculation of liquid heat capacities using the Rowlinson/Bondi correlation method. This function can be invoked as

```
cpL = hcRB(T, Tc, w, Cpi)
```

where T is the temperature (K), T_c is the critical temperature (K), w is the acentric factor, Cpi is the ideal gas heat capacity (J/(mol·K)), and cpL is the estimated liquid heat capacity (J/(mol·K)).

```
function cpL = hcRB(T, Tc, w, Cpi)
% Estimation of liquid heat capacity by Rowlinson/Bondi method
% input:
%   T, Tc: temperature and critical temperature (K)
%   w: acentric factor
%   Cpi: ideal gas heat capacity (J/mol/K)
% output
%   cpL: estimated liquid heat capacity (J/mol/K)

Tr = T./Tc; R = 8.3143;
cpL = Cpi + 1.45*R + 0.45*R./(1-Tr) + 0.25*w*R*(17.11 +...
    25.2*(1-Tr).^1/3 ./ Tr + 1.742./(1-Tr)); % (J/mol/K)
end
```

Example 3.8: Heat Capacity of MEK

Use the Rowlinson/Bondi method to estimate the specific heat capacity of methyl ethyl ketone (MEK) at $T = 100^\circ\text{C}$. Use $C_p^{id} = 1.671\text{ J/(g}\cdot\text{K)} = 120.496\text{ J/(mol}\cdot\text{K)}$, $T_c = 535.55\text{ K}$, $w = 0.323$, $M_w = 72.11\text{ g/mol}$, and $R = 8.3143\text{ J/(mol}\cdot\text{K)}$.

Solution

```
>> T = 373.15; Tc = 535.55; w = 0.323; Cpi = 120.496;
>> cpL = hcRB(T, Tc, w, Cpi)
cpL =
    162.6885
```

3.5.2 HEAT CAPACITIES OF GASES

The heat capacity of an ideal gas C_p^0 at low pressure can be expressed as a third-degree polynomial, which is a function of temperature:

$$C_p^0 = A + BT + CT^2 + DT^3$$

where

C_p^0 is the heat capacity of the ideal gas at low pressure (cal/(gmol·K))

T is the temperature (K)

A, B, C , and D are correlation constants

The MATLAB function *hcapG* calculates heat capacities of gases listed in [Table 3.3](#). This function defines constants *A*, *B*, *C*, and *D* in the range of $298 \text{ K} \leq T \leq 1500 \text{ K}$. This function can be used as

```
hcp = hcapG(T, cname)
```

where *T* is the temperature (K); *cname* is the name of the component, which can be specified as a chemical formula or common name; and *hcp* is the resultant heat capacity of the gas (cal/(gmol·K)). *T* can be either a scalar or a row vector representing a temperature range.

```
function hcp = hcapG(T, cname)
% Calculates gas heat capacity (cal/gmol/K)
% input
%   T: temperature(K) (scalar or a row vector)
%   cname: common name or chemical formula of the compound
% output
% hcp: gas heat capacity (cal/gmol/K)

% correlation constants (A, B, C, D)
cv = [5.8900    6.9400   -5.4800    1.5200
      7.0000    5.0500   -4.3900    1.3000
      5.8500   15.4000  -11.1000   2.9100
      6.9200   -0.6500    2.8000   -1.1400
      5.1400   15.4000  -9.9400    2.4200
      7.2400   -1.7600    3.0700   -1.0000
      6.0700    8.2300   -0.1600   -0.6600
      8.1000   -0.7200    3.6300   -1.1600
      5.5200   19.8000  -13.9000   3.7400
      6.8800   -0.0220    0.2100   0.1300
      7.0700   -1.3200    3.3100   -1.2600
      6.2200    2.7100   -0.3700   -0.2200
      0.9340   36.9000  -19.3000   4.0100
      5.0400    9.3200    8.8700   -5.3700
      2.4600   36.1000   -7.0000   -0.4600
      -0.5800   69.9000  -32.9000   6.5400
      -8.7900  116.0000  -76.1000  18.9000
      -9.3400  138.5000  -87.2000  20.6000
      -8.1100  143.4000 -107.0000  30.7000
      -5.6800  126.0000  -85.4000  20.5000
      -6.2800   79.8000  -50.5000  12.2000
      0.0      0.0      0.0      0.0 % missing component (C6H12)
      -0.5600   81.1000  -53.5000  13.6000
      3.6200   24.9000   -7.0500    0
      7.7100   34.3000  -26.4000   7.2900
      12.6000   33.2000  -29.3000   8.5800];
wv = [cv(:,1) cv(:,2)*1e-3 cv(:,3)*1e-6 cv(:,4)*1e-9];
ind = compID(cname); % identification number of the component
% computes gas heat capacity
hcp = wv(ind,1) + wv(ind,2).*T + wv(ind,3).*T.^2 + wv(ind,4).*T.^3;
end
```

Example 3.9: Heat Capacity of Water

Estimate the heat capacity of water at 150°C.

Solution

```
>> T = 150+273.15;
>> v = hcapL(T, 'Water')
v =
    1.0223
```

Example 3.10: Heat Capacity of Carbon Dioxide

Estimate the heat capacity of carbon dioxide at 300°C.

Solution

```
>> T = 300+273.15;
>> v = hcapG(T, 'CO2')
v =
    11.1568
```

3.6 THERMAL CONDUCTIVITY

3.6.1 THERMAL CONDUCTIVITIES OF LIQUIDS

The thermal conductivity of saturated liquids can be estimated by

$$k_L = A + BT + CT^2$$

where

- k_L is the thermal conductivity of a saturated liquid ($\mu\text{cal}/(\text{s} \cdot \text{cm} \cdot ^\circ\text{C})$)
- T is the temperature (K)
- A , B , and C are correlation constants

Values of k_L for most common organic liquids range between 250 and 400 $\mu\text{cal}/(\text{s} \cdot \text{cm} \cdot ^\circ\text{C})$ at temperatures below the boiling point. The MATLAB function *condL* calculates thermal conductivities of liquid components listed in [Table 3.3](#). This function defines constants A , B , and C . This function has the syntax

```
k = condL(T, cname)
```

where T is the temperature (K); $cname$ is the name of the component, which can be specified as a chemical formula or common name; and k is the estimated thermal conductivity of the specified liquid ($\mu\text{cal}/(\text{s} \cdot \text{cm} \cdot ^\circ\text{C})$). T can be either a scalar or a row vector representing a temperature range.

```
function k = condL(T, cname)
% Calculates liquid thermal conductivity (microcal/cm/s/C)
% input
%   T: temperature(K) (scalar or a row vector)
%   cname: common name or chemical formula of the compound
% output
%   k: liquid thermal conductivity (microcal/cm/s/C)

% correlation constants (A, B, C)
cv = 1000*[0.6215 -0.1623 -0.1184
            0.5590 -0.0483 -0.0152
            2.1408 -0.7837 0.0714
            0.4755 0.0033 -0.2143
            0.9721 -0.2015 -0.0230
            1.0717 -0.0184 -0.0658
            2.5513 -0.3766 -0.0294
            -0.9166 1.2547 -0.1521
            -0.4666 0.8059 -0.0876
            -0.0201 2.4737 -5.3473
            0.6280 -0.3689 -0.0226
```

```

0.5838 -0.2105 -0.0483
0.8515 -0.2289 -0.0047
0.7227 -0.1444 -0.0764
0.6993 -0.1659 -0.0049
0.6235 -0.1268 -0.0021
0.4243 0.0011 -0.0090
0.4851 -0.0538 -0.0006
0.5375 -0.0304 -0.0015
0.4409 0.0867 -0.0070
0.3968 -0.0421 -0.0067
0.3883 -0.0227 -0.0033
0.7183 -0.1872 0.0117
0.7701 -0.1142 0.0028
0.3902 -0.0206 -0.0051];
wv = [cv(:,1) cv(:,2)*1e-2 cv(:,3)*1e-4];
ind = compID(cname); % identification number of the component
% Computes liquid thermal conductivity
k = wv(ind,1) + wv(ind,2).*T + wv(ind,3).*T.^2;
end

```

3.6.2 THERMAL CONDUCTIVITIES OF GASES

The thermal conductivities of low-pressure gases increase with temperature. The thermal conductivity k_G of a gas can be correlated as a function of temperature as

$$k_G = A + BT + CT^2 + DT^3$$

where

k_G is the thermal conductivity of the gas at low pressure ($\mu\text{cal}/(\text{cm} \cdot \text{s} \cdot \text{K})$)

T is the temperature (K)

A, B, C , and D are correlation constants

The MATLAB function *condG* calculates thermal conductivities of gas components listed in [Table 3.3](#). This function defines constants A, B, C , and D . A simple representation of its syntax is

```
k = condG(T, cname)
```

where T is the temperature (K); $cname$ is the name of the gas component, which can be specified as a chemical formula or common name; and k is the estimated thermal conductivity of the specified gas ($\mu\text{cal}/(\text{s} \cdot \text{cm} \cdot ^\circ\text{C})$). T can be either a scalar or a row vector representing a temperature range.

```

function k = condG(T, cname)
% Calculates thermal conductivities of gases (microcal/cm/s/K)
% input
%   T: temperature(K) (scalar or a row vector)
%   cname: common name or chemical formula of the compound
% output
%   k: bas thermal conductivity (microcal/cm/s/K)

% correlation constants (A, B, C, D)
cv = [1.8654 19.7900 1.2400 -17.7700
      3.2500 5.8000 0.2100 -1.2500
     -19.3100 15.1500 -0.3300 0.5500

```

```

1.2100  21.7900  -0.8416  1.9580
-17.2300 19.1400  0.1308  -2.5140
-0.2600 12.6700  -0.2500  0.1600
0.9100 12.8700  2.9300  -8.6800
17.5300 -2.4200  4.3000  -21.7300
-21.0700 16.9700  0.1700  -1.5600
19.3400 159.7400 -9.9300  37.2900
0.9359 23.4400  -1.2100  3.5910
-0.7816 23.8000  -0.8939  2.3240
-42.0400 28.6500  0.7963  -3.2620
-4.4630 20.8400  2.8150  -8.6310
-75.8000 52.5700 -4.5930  39.7400
4.4380 -1.1220  5.1980  -20.0800
-20.1900 8.6400  2.3400  -9.6900
18.1400 -9.5700  5.6600  -22.2200
-26.3900 11.8900  1.5500  -4.3000
-31.8700 15.2600  1.7400  -4.4000
-20.4600 9.7400  3.7700  -16.2800
-20.5700 4.4500  4.0700  -17.3100
-67.9200 29.9600  1.7400  -12.2000
-18.6200 9.9500  2.9000  -12.3800
-5.7320 6.2900  0.5904  -3.3520
-0.4161 4.0670  0.6115  -3.5660];
wv = [cv(:,1) cv(:,2)*1e-2 cv(:,3)*1e-4 cv(:,4)*1e-8];
ind = compID(cname); % % identification number of the component
% Computes gas thermal conductivity
k = wv(ind,1) + wv(ind,2)*T + wv(ind,3)*T.^2 + wv(ind,4)*T.^3;
end

```

Example 3.11: Thermal Conductivity of Water

Calculate the thermal conductivity of water at the range of $0^{\circ}\text{C} \leq T \leq 350^{\circ}\text{C}$, and plot the results as a function of temperature.

Solution

The following commands compute the thermal conductivity and plot the results as shown in [Figure 3.2](#):

```

>> T = [0:350]+273.15; k = condL(T,'water');
>> plot(T-273.15, k), xlabel('T(C)')
>> ylabel('Thermal conductivity of water(\mu\text{cal/s/cm/C})'), grid on

```

Example 3.12: Thermal Conductivity of Propane

Estimate the thermal conductivity of propane gas at the range of $25^{\circ}\text{C} \leq T \leq 900^{\circ}\text{C}$, and plot the results as a function of temperature.

Solution

The following commands generate a plot of the results as shown in [Figure 3.3](#).

```

>> T = [25:900]+273.15; k = condG(T,'propane');
>> plot(T-273.15, k), xlabel('T(C)')
>> ylabel('Thermal conductivity of propane(\mu\text{cal/s/cm/K})')
>> axis tight, grid on

```

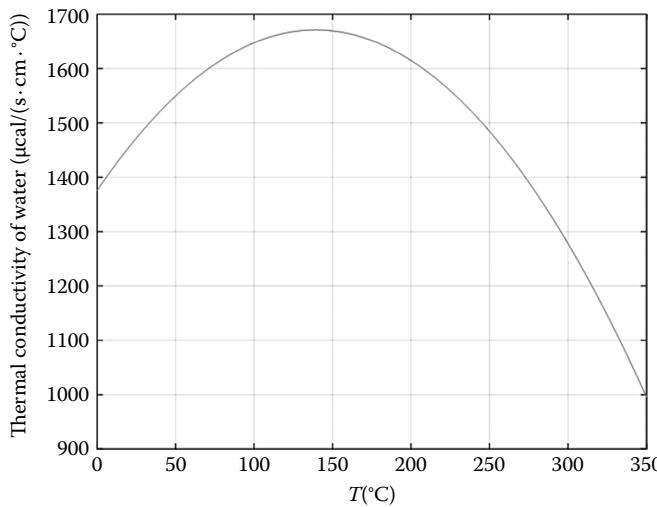


FIGURE 3.2 Plot of the thermal conductivity of water.

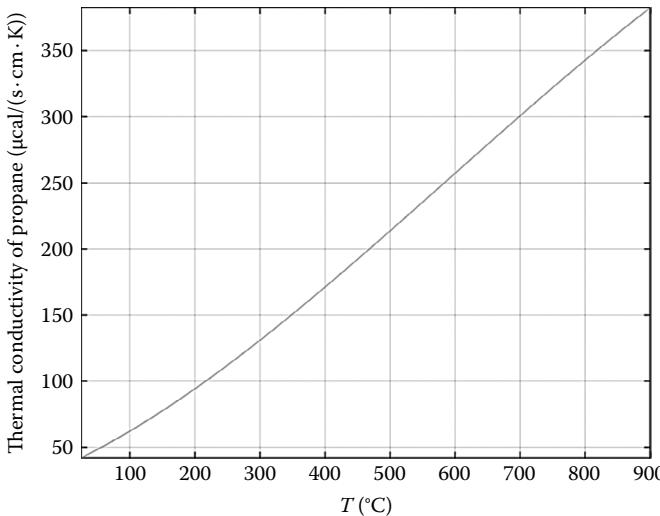


FIGURE 3.3 Plot of the thermal conductivity of propane.

3.7 SURFACE TENSION

Surface tensions of liquids can be expressed as¹³

$$\sigma = \sigma_1 \left(\frac{T_c - T}{T_c - T_1} \right)^r \quad \text{or} \quad \frac{\sigma}{\sigma_1} = \left(\frac{T_c - T}{T_c - T_1} \right)^r$$

where

σ_1 is the surface tension (dynes/cm) at T_1 (K)

T_c is the critical temperature (K)

r is a correlation parameter

For water, the temperature range for which the surface tension is valid is

$$0^\circ\text{C} - 100^\circ\text{C}: T_1 = 298.16 \text{ K}, \quad r = 0.8105, \quad \sigma_1 = 71.97$$

$$100^\circ\text{C} - 374.2^\circ\text{C}: T_1 = 373.16 \text{ K}, \quad r = 1.1690, \quad \sigma_1 = 58.91$$

The MATLAB function *surL* calculates surface tensions of components listed in [Table 3.3](#). This function defines the value of *r*. Its basic syntax is

```
sg = surtL(T, cname)
```

where *T* is the temperature (C); *cname* is the name of the component, which can be specified as a chemical formula or common name; and *sg* is the ratio of the estimated surface tension to the surface tension at *T*₁. If *cname* is water, *sg* is the value of the estimated surface tension (dynes/cm).

```
function sg = surtL(T, cname)
% Calculates liquid surface tension (dynes/cm)
% input
% T: temperature(C) (scalar or a row vector)
% cname: common name or chemical formula of the compound
% output
% sg: ratio of surface tension at T to that at T1 (sg = sigma/sigma1)

% data (T1, Tc) and correlation constant (r)
wv = [ -200.0000 -129.0000 0.8811
        20.0000 144.0000 1.0508
        30.0000 157.6000 1.1768
       -193.0000 -140.1000 1.1441
        20.0000 31.1000 1.3015
       -93.0000 51.5000 1.0972
       -45.0000 132.4000 1.1548
       25.0000 374.2000 0.8105
       18.2000 455.0000 0.9141
      -256.0000 -240.2000 1.1012
      -203.0000 -146.8000 1.2123
      -202.0000 -118.5000 1.1933
      -120.0000 9.9000 1.2760
      -168.1600 -82.6000 1.3941
      -120.0000 32.3000 1.2060
       -90.0000 96.7000 1.1982
       20.0000 288.9400 1.2243
       20.0000 318.8000 1.2364
       20.0000 426.0000 1.1022
       60.0000 420.0000 1.0725
      10.0000 124.9000 1.3201
      20.0000 280.3000 1.4246
      40.0000 152.0000 1.2055
      20.0000 239.4000 0.8115
      25.0000 263.4000 1.1824
      30.0000 283.2000 1.2278];
ind = compID(cname); % identification number of the component
% compute ratio of surface tension
T0 = 273.15; T = T + T0; T1 = wv(ind,1) + T0; Tc = wv(ind,2) + T0;
if ind ~= 8
    sg = ((Tc - T) ./ (Tc - T1)) .^ (wv(ind,3));
else
    sg = 1;
end
```

```

else % ind=8: water
    sg = [] ;
    for j = 1:length(T)
        if T(j) >= T0 && T(j)-T0 <= 100+T0 % 0<=T<=100 (C)
            sg1 = 71.97;
            sgv = sg1*((wv(ind,2) + T0 - T(j))./(wv(ind,2) + ...
                T0 - (wv(ind,1) + T0))).^(wv(ind,3));
        else % 100 <= T(C) <= 374.2
            wv(ind,:) = [100 374.2 1.169]; sg1 = 58.91;
            sgv = sg1*((wv(ind,2) + T0 - T(j))./(wv(ind,2) + ...
                T0 - (wv(ind,1) + T0))).^(wv(ind,3));
        end
        sg = [sg sgv];
    end
end

```

For example, the surface tension of benzene at 60°C is given by

```

>> surtL(60, 'benzene')
ans =
0.8211

```

For cryogenic liquids, the surface tension can be estimated by the equation proposed by Sprows and Prausnitz¹⁴:

$$\sigma = \sigma_0 (1 - T_r)^p$$

In this equation, σ_0 and p can be determined by the least-squares analysis of the measured data.

For nonpolar liquids, the corresponding states correlation can be used to estimate surface tensions¹⁵:

$$\sigma = P_c^{2/3} T_c^{1/3} Q (1 - T_r)^{11/9}$$

$$Q = 0.1196 \left(1 + \frac{T_{br} \ln(P_c / 1.03125)}{1 - T_{br}} \right) - 0.279$$

where

P_c is the critical pressure (bar)

T_c is the critical temperature (K)

T_b is the normal boiling point (K), and $T_{br} = T_b/T_c$

Pitzer proposed a series of relations for σ in terms of P_c , T_c , and ω that together led to the following corresponding states relation for σ ¹⁶:

$$\sigma = P_c^{2/3} T_c^{1/3} \frac{1.86 + 1.18\omega}{19.05} \left(\frac{3.75 + 0.91\omega}{0.291 - 0.08\omega} \right)^{2/3} (1 - T_r)^{11/9}$$

where ω is the acentric factor.

The MATLAB function *corstsg* calculates the surface tension using the corresponding states correlation, and the function *pitzersg* uses Pitzer's relation to estimate the surface tension. The basic syntax is

```
sg = corstsg(Pc,Tc,Tb,T)
sg = pitzersg(w,Pc,Tc,T)
```

where P_c is the critical pressure (bar), T_c is the critical temperature (K), T_b is the normal boiling point (K), w is the acentric factor, T is the temperature (K), and sg is the estimated surface tension (dyne/cm).

corstsg.m

```
function sg = corstsg(Pc,Tc,Tb,T)
% Estimation of surface tension using the corresponding states
% correlation
% input
% Pc: critical pressure (bar)
% Tc: critical temperature (K)
% Tb: normal boiling point (K)
% T: temperature (K)
% output
% sg: surface tension (dyne/cm)

Tr = T./Tc; Tbr = Tb./Tc;
Q = 0.1196*(1 + Tbr.*log(Pc/1.01325)./(1-Tbr)) - 0.279;
sg = Pc.^(2/3).*Tc.^(1/3).*Q.*^(1-Tr).^(11/9);
end
```

pitzersg.m

```
function sg = pitzersg(w,Pc,Tc,T)
% Estimation of surface tension using Pitzer's relation
% input
% Pc: critical pressure (bar)
% Tc: critical temperature (K)
% w: acentric factor
% T: temperature (K)
% output
% sg: surface tension (dyne/cm)

Tr = T./Tc;
sg = (Pc.^(2/3)).*(Tc.^(1/3)).*((1.86+1.18*w)/19.05).*((3.75+...
0.91*w)./(0.291-0.08*w)).^(2/3).*^(1-Tr).^(11/9);
end
```

Example 3.13: Surface Tension of Ethanethiol

Use the corresponding states correlation to estimate the surface tension of ethanethiol (ethyl mercaptan) at 30°C. For ethanethiol, $P_c = 54.9$ bar, $T_c = 499$ K and $T_b = 308.15$ K.

Solution

The function *corstsg* is used to evaluate the surface tension:

```
>> T = 303.15; Tc = 499; Tb = 308.15; Pc = 54.9;
>> sg = corstsg(Pc,Tc,Tb,T)
sg =
22.3398
```

3.8 VAPOR PRESSURE

3.8.1 ANTOINE EQUATION

The Antoine equation is often used to estimate the vapor pressure P_v :

$$\ln P_v = A - \frac{B}{T+C}$$

This equation is applicable for pressures with ranges from 10 to 1500 mmHg.

3.8.1.1 Extended Antoine Equation

The vapor pressure of the saturated liquid can be expressed as a function of temperature¹⁷:

$$\log P_v = A + \frac{B}{T} + C \log T + DT + ET^2$$

where

P_v is the vapor pressure of the saturated liquid (mmHg)

T is the temperature (K)

A, B, C, D , and E are correlation constants

The MATLAB function *prVp* calculates the vapor pressure of components listed in [Table 3.3](#). This function defines correlation constants A, B, C, D , and E . The basic syntax is

```
pv = prVp(T, cname)
```

where T is the temperature (C); $cname$ is the name of the component, which can be specified either as a chemical formula or a common name; and pv is the estimated vapor pressure (mmHg) at T .

```
function pv = prVp(T, cname)
% Estimation of vapor pressure of the saturated liquid (mmHg)
% input
% T: temperature(C) (scalar or a row vector)
% cname: common name or chemical formula of the compound
% output
% pv: vapor pressure(mmHg) at T

% identification number of the component
ind = compID(cname);

% correlation constants (A, B, C, D, E)
cv = 1.0e+04 * [
  0.0021480 -0.051651 -0.00071218 0.00143550 0
  0.0042262 -0.20098 -0.001396300 0.00093705 0
  0.0046554 -0.24563 -0.001516900 0.00090026 0
  0.0032863 -0.06069 -0.001296900 0.00275510 0
  0.0047544 -0.17922 -0.001655900 0.00138330 0
  0.0136050 -0.30473 -0.005841600 0.00954960 -0.0058507
  0.0038440 -0.20662 -0.001210500 0.00077768 0
  0.0016373 -0.28186 -0.000169080 -0.00057546 0.0004007
```

```

0.0044791 -0.40227 -0.001307600 0.00045627 0
0.0005237 -0.00463 -0.000044809 0.00252900 0
0.0021623 -0.04556 -0.000751070 0.00172140 0
0.0005648 -0.04113 0.000181180 -0.00250420 0.0062610
0.0030895 -0.11968 -0.001015300 0.00099351 0
0.0022573 -0.06562 -0.000739420 0.00118960 0
0.0016316 -0.10748 -0.000314340 0.00045534 0.0010373
0.0036007 -0.17372 -0.001166600 0.00085187 0
0.0051204 -0.32457 -0.001640300 0.00075400 0
0.0115210 -0.49181 -0.004346700 0.00385480 -0.0013496
0.0018893 -0.31040 -0.000347140 0.00000274 0
0.0672710 -2.21973 -0.026667000 0.02264300 -0.0073731
0.0038450 -0.18652 -0.001257800 0.00089375 0
0.0064753 -0.36192 -0.002175300 0.00107420 0
0.0041401 -0.21812 -0.001345100 0.00084524 0
-0.0042629 -0.11862 0.002327900 -0.00350820 0.00175780
0.0026828 -0.22926 -0.000718600 0.00031365 0
0.0050612 -0.31357 -0.001631300 0.00078036 0];
wv = [cv(:,1) cv(:,2) cv(:,3) cv(:,4)*1e-3 cv(:,5)*1e-6];

% compute vapor pressure
T0 = 273.15; T = T + T0;
logp = wv(ind,1) + wv(ind,2)./T + wv(ind,3).*log10(T) +...
        wv(ind,4).*T + wv(ind,5).*T.^2;
pv = 10.^logp;
end

```

The extended Antoine equation shown below can be used to estimate the vapor pressure at high temperatures¹⁸:

$$\log_{10} P_{vb} = A - \frac{B}{T + C - 273.15} + 0.43429x^n + Ex^8 + Fx^{12}$$

where

P_{vb} is the vapor pressure (bar)

T is the temperature (K)

$x = (T - t_0 - 273.15)/T_c$

A, B, C, n, E, F , and t_0 are constants

Example 3.14: Vapor Pressure of Water

Estimate the vapor pressure of water at the range of $0^\circ\text{C} \leq T \leq 350^\circ\text{C}$, and plot the results as a function of temperature.

Solution

The following code produces the plot shown in [Figure 3.4](#).

```

>> T = 0:350;
>> pv = prVp(T, 'H2O');
>> plot(T,pv), xlabel('T(C)'), ylabel('Vapor pressure of water(mmHg)'), grid on

```

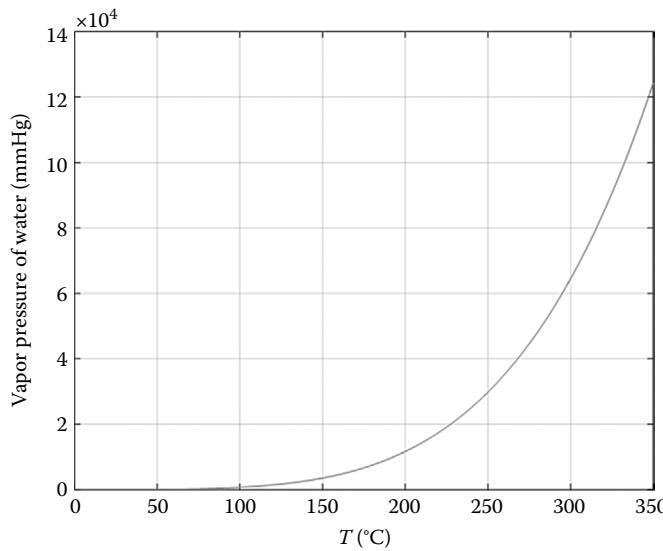


FIGURE 3.4 Vapor pressure of water.

3.8.2 WAGNER EQUATION

The Wagner equation is a very useful correlation to estimate vapor pressures.¹⁹ This equation correlates the whole vapor pressure curve from the triple point to the critical point.

$$\ln \frac{P_v}{P_c} = \frac{1}{T_r} \left[A(1-T_r) + B(1-T_r)^{1.5} + C(1-T_r)^{2.5} + D(1-T_r)^5 \right]$$

where $T_r = T/T_c$. Reasonable ranges of parameters are $-9 \leq A \leq -5$, $-10 \leq B \leq 10$, $-10 \leq C \leq 10$, and $-20 \leq D \leq 20$. The MATLAB function *vpwagner* uses the Wagner equation to estimate vapor pressure. The basic syntax is

```
Pv = vpwagner(T, Tc, Pc, C)
```

where T is the temperature (K), Tc is the critical temperature (K), Pc is the critical pressure (MPa), C is the parameter vector, and Pv is the estimated vapor pressure (MPa).

```
function Pv = vpwagner(T, Tc, Pc, C)
% Estimation of vapor pressure using the Wagner equation
% input:
% T, Tc: temperature (K) and critical temperature (K)
% Pc: critical pressure (MPa)
% C: parameter vector of the Wagner equation
% output
% Pv: estimated vapor pressure (MPa)

Tr = T./Tc; a = C(1); b = C(2); c = C(3); d = C(4);
Pv = Pc.*exp((a*(1-Tr) + b*(1-Tr).^1.5 + c*(1-Tr).^2.5 + d*(1-Tr).^5)/Tr);
end
```

Example 3.15: Vapor Pressure of Acetone

Estimate the vapor pressure (MPa) of acetone at 273.15 K. For acetone, $T_c = 508.1$ K, $P_c = 4.6924$ MPa, and values of parameters of the Wagner equation are $A = -7.670734$, $B = 1.965917$, $C = -2.445437$, and $D = -2.899873$.

Solution

```
>> C = [-7.670734 1.965917 -2.445437 -2.899873];
>> T=273.15; Tc=508.1; Pc=4.6924;
>> Pv = vpwagner(T,Tc,Pc,C)
Pv =
0.0093
```

3.8.3 HOFFMANN–FLORIN EQUATION

The vapor pressure can be calculated by a simple estimation procedure based on the equation of Hoffmann and Florin.²⁰ The Hoffmann–Florin equation has two adjustable parameters α and β .

$$\ln \frac{P_v}{P_0} = \alpha + \beta f(T)$$

where

$$f = \frac{1}{T} - 7.9151 \times 10^{-3} + 2.6726 \times 10^{-3} \log(T) - 0.8625 \times 10^{-6} T \quad (T : \text{K})$$

$$\alpha = \ln \frac{P_1^v}{P_0} - \ln \frac{P_1^v}{P_2^v} \cdot \frac{f(T_1)}{f(T_1) - f(T_2)}, \quad \beta = \ln \frac{P_1^v / P_2^v}{f(T_1) - f(T_2)}$$

3.8.4 RAREY/MOLLER EQUATION

The Rarey/Moller equation is based on a new group contribution estimation method with the normal boiling point as Reference 21. In this equation, the C parameter of the Antoine equation is correlated with the normal boiling point.

$$\begin{aligned} \ln P_v &= B' \left(\frac{T - T_b}{T + 2.65 - (T_b^{1.485} / 135)} \right) + D' \ln \frac{T}{T_b} \\ B' &= 9.42208 + \sum_i v_i \Delta B_i + n_A \sum_j v_j \Delta B_j + \sum_k \Delta B_k + \frac{1}{2} \sum_i \sum_j GI_{ij} \\ D' &= D + \frac{1}{n_A} \sum_i v_i \Delta E_i \end{aligned}$$

where

P_v is the vapor pressure (bar)

T is the temperature (K)

ΔB_i are the group contributions of the structural groups

v_i is the frequency of group i in the molecule

The GI_{ij} are the group interaction contributions with $G_{ij} = G_{ji}$, D' denotes a correction term for aliphatic alcohols and carboxylic acids consisting of group contributions ΔD_i and a constant D , and n_A is the number of atoms except hydrogen.

The MATLAB function *vpRM* performs the vapor pressure estimation by using the Rarey/Moller equation. This function can be called as

```
Pv = vpRM(T, Tb, nu, dB, GI, Dp)
```

where T and T_b are temperature and normal boiling point (K), respectively; nu is the frequency of groups; dB and GI denote ΔB_i and GI_{ij} of each group, respectively; Dp is D' ; and Pv is the estimated vapor pressure (bar).

```
function Pv = vpRM(T, Tb, nu, dB, GI, Dp)
% Estimation of vapor pressure by using the Rarey/Moller equation
% input
% T, Tb: temperature and normal boiling point (K)
% nu: frequency of groups
% dB, GI: delta B and GIij of each group
% Dp: D prime
% output
% Pv: vapor pressure (bar)
sumdBi = sum(nu.*dB); sumGI = sum(sum(GI));
Bp = 9.42208 + sumdBi + sumGI;
Pv = exp(Bp*(T-Tb)./(T+2.65-(Tb.^1.485)/135) + Dp*log(T./Tb)); % bar
end
```

Example 3.16: Vapor Pressure of *n*-Propyl Benzene²²

Estimate the vapor pressure of *n*-propyl benzene ($C(CH_3)_2(CH_2)_2CH_3$) at 100°C and 200°C by using the Rarey/Moller equation. The normal boiling point of *n*-propyl benzene is $T_b = 159.22^\circ C$. Each group and corresponding frequency of *n*-propyl benzene molecule are shown in Table 3.4.

Solution

The function *vpRM* returns vapor pressure for each temperature.

```
>> Tb = 159.22+273.15; nu = [2 1 1 5]; Dp = 0; GI = [];
>> dB = [0.07545 -0.00227 0.11192 0.01653];
>> T = 100 + 273.15; Pv = vpRM(T, Tb, nu, dB, GI, Dp)
Pv =
    0.1595
>> T = 200 + 273.15; Pv = vpRM(T, Tb, nu, dB, GI, Dp)
Pv =
    2.6106
```

TABLE 3.4
Groups and Frequency

Group	v_i (Frequency)	ΔB_i
CH_2	2	0.07545
CH_3	1	-0.00227
C	1	0.11192
CH	5	0.01653

3.9 ENTHALPY OF VAPORIZATION

3.9.1 WATSON EQUATION

The enthalpy of vaporization, ΔH_v , is also termed the latent heat of vaporization. ΔH_v is the difference between the enthalpy of the saturated vapor and that of saturated liquid at the same temperature. At a given temperature T (K), ΔH_v can be estimated by the Watson correlation²³:

$$\Delta H_v = \Delta H_{v1} \left(\frac{T_c - T}{T_c - T_1} \right)^r$$

where

ΔH_v is the heat of vaporization (cal/g) at a given temperature

ΔH_{v1} is the heat of vaporization (cal/g) at T_1 (K)

T_c is the critical temperature (K)

r is the characteristic constant for the component (usually $r = 0.38$)

The MATLAB function *hvapn* calculates the enthalpy of vaporization of components listed in Table 3.3. This function defines ΔH_{v1} , T_1 , T_c , and r . The basic syntax is

```
hv = hvapn(T, cname)
```

where T is the temperature (C); $cname$ is the name of the component, which can be specified as a chemical formula or common name; and hv is the estimated heat of vaporization (cal/g) at T .

```
function hv = hvapn(T, cname)
% Estimation of heat of vaporization (cal/g)
% input
% T: temperature(K) (scalar or a row vector)
% cname: common name or chemical formula of the compound
% output
% hv: estimated heat of vaporization at T (cal/g)
% parameters (dhv1, T1(C), Tc(C), r)
cv = [41.1000 -188.1000 -129.000 0.3800
       69.6000 -34.0600 144.0000 0.3800
       93.0000 -10.0000 157.6000 0.3800
       51.6000 -191.5000 -140.1000 0.3800
       56.1000 0 31.1000 0.3800
       105.8000 -85.0300 51.5000 0.3800
       327.4000 -33.4300 132.4000 0.3800
       538.7000 100.0000 374.2000 0.3800
       321.9000 150.2000 455.0000 0.3800
       107.0000 -252.8000 -240.2000 0.2370
       47.5000 -195.8000 -146.8000 0.3800
       50.9000 -183.0000 -118.5000 0.3800
       115.4000 -103.7000 9.9000 0.3800
       121.7000 -161.5000 -82.6000 0.3800
       116.7000 -88.2000 32.3000 0.3800
       101.8000 -42.1000 96.7000 0.3800
       94.1000 80.1000 288.9400 0.3800
       86.1000 110.6000 318.8000 0.3800
       112.4000 184.4000 426.0000 0.3800
       116.4000 181.8000 420.0000 0.3800
       113.8000 -32.8000 124.9000 0.3800
```

```

0.0 0.0 0.0 0.0 0.0 % missing component (C6H12)
100.2000 -4.4100 152.0000 0.3800
260.1000 64.7000 239.4000 0.4000
58.9000 61.3000 263.4000 0.3800
46.5500 76.7000 283.2000 0.3800];
ind = compID(cname); % identification number of the component
% computes heat of vaporization
T0 = 273.15;
T = T + T0; Tc = cv(ind,3) + T0; T1 = cv(ind,2) + T0;
hv = cv(ind,1)*((Tc - T)./(Tc - T1)).^(cv(ind,4));
end

```

Example 3.17: Heat of Vaporization of Water

Estimate the heat of vaporization of water (cal/g) at the range of $0^\circ\text{C} \leq T \leq 350^\circ\text{C}$, and plot the result as a function of temperature.

Solution

The following code produces the plot shown in Figure 3.5.

```

>> T = 0:350;
>> hv = hvapn(T, 'H2O');
>> plot(T,hv), xlabel('T(C)'), ylabel('Heat of vaporization of water(cal/g)'), grid on

```

3.9.2 PITZER CORRELATION

The enthalpy of vaporization, ΔH_v , can be estimated by the correlation proposed by Pitzer et al.²⁴:

$$\frac{\Delta H_v}{RT_c} = 7.08(1-T_r)^{0.354} + 10.95\omega(1-T_r)^{0.456}$$

where ω is the acentric factor.

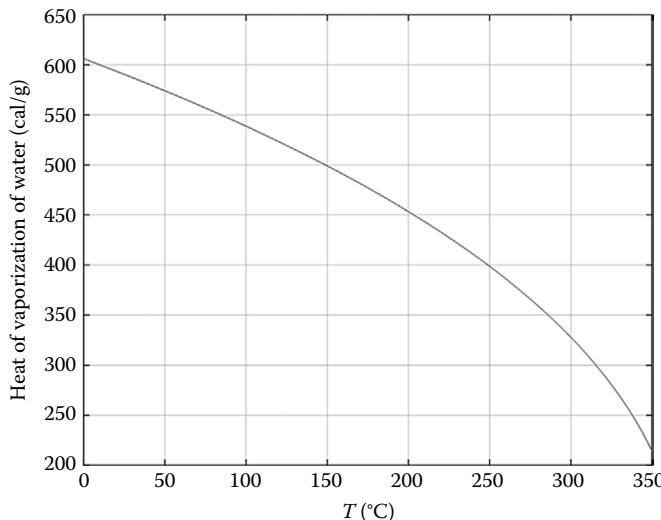


FIGURE 3.5 Heat of vaporization of water as a function of temperature.

3.9.3 CLAUSIUS–CLAPEYRON EQUATION

The derivative of the vapor pressure with respect to temperature in the Clausius–Clapeyron equation can be replaced with a suitable relation to be used in the estimation of the enthalpy of vaporization:

$$\Delta H_V = -R \frac{d \ln P_v}{d(1/T)} = T \left(v^V - v^L \right) \frac{dP_v}{dT} \quad (T_r < 0.75)$$

Use of the Antoine equation: If the Antoine equation $\log P_v = A - (B/(T + C))$ is used, the derivative of the vapor pressure with respect to temperature can be given by

$$\frac{d \ln P_v}{d(1/T)} = - \frac{2.3026B(T + 273.15)^2}{(T + C)^2}$$

Therefore, we have

$$\Delta H_V = \frac{2.3026RB(T + 273.15)^2}{(T + C)^2}$$

where

T is the temperature ($^{\circ}\text{C}$)

B and C are parameters of the Antoine equation

Use of the Wagner equation: The Wagner correlation can be used to evaluate the derivative dP_v/dT . Differentiation of the Wagner equation

$$\ln \frac{P_v}{P_c} = \frac{1}{T_r} \left[A(1 - T_r) + B(1 - T_r)^{1.5} + C(1 - T_r)^{2.5} + D(1 - T_r)^5 \right]$$

with respect to T gives²⁵

$$\frac{dP_v}{dT} = -\frac{P_v}{T} \left[\ln \frac{P_v}{P_c} + A + 1.5B(1 - T_r)^{0.5} + 2.5C(1 - T_r)^{1.5} + 5D(1 - T_r)^4 \right]$$

Substitution of this equation into the Clausius–Clapeyron equation yields

$$\Delta H_V = -P_v \left(v^V - v^L \right) \left[\ln \frac{P_v}{P_c} + A + 1.5B(1 - T_r)^{0.5} + 2.5C(1 - T_r)^{1.5} + 5D(1 - T_r)^4 \right]$$

The MATLAB function *vhwagner* estimates the enthalpy of vaporization by using the Wagner equation. The syntax for this function is

`dHv = vhwagner(T, Tc, Pc, vL, vV, C)`

where T is the temperature (K), T_c is the critical temperature (K), P_c is the critical pressure (MPa), C is the parameter vector of the Wagner equation, and dHv is the estimated enthalpy of vaporization (J/mol).

```

function dHv = vh Wagner(T,Tc,Pc,vL,vV,C)
% Estimation of enthalpy of vaporization using the Wagner equation
% input:
% T,Tc: temperature and critical temperature (K)
% Pc: critical pressure (MPa)
% C: parameter vector of the Wagner equation
% output
% dHv: estimated enthalpy of vaporization (J/mol)
Tr = T./Tc; a = C(1); b = C(2); c = C(3); d = C(4);
Pv = Pc.*exp((a*(1-Tr) + b*(1-Tr).^1.5 + c*(1-Tr).^2.5 +
d*(1-Tr).^5)/Tr);
wd = log(Pv./Pc) + a + 1.5*b*(1-Tr).^0.5 + 2.5*c*(1-Tr).^1.5 +
5*d*(1-Tr).^4;
dHv = -Pv*(vV - vL).*wd*1e6; % J/mol
end

```

Example 3.18: Enthalpy of Vaporization of Acetone

Estimate the enthalpy of vaporization of acetone at $T = 273.15$ K (0°C). For acetone, $v^L = 7.145 \times 10^{-5}$ m³/mol, $v^V = 0.2453$ m³/mol, $T_c = 508.1$ K, $P_c = 4.6924$ MPa, $M_w = 58.08$ g/mol, and the parameters of the Wagner equation are $A = -7.670734$, $B = 1.965917$, $C = -2.445437$, and $D = -2.899873$.

Solution

```

>> C = [-7.670734 1.965917 -2.445437 -2.899873];
>> T=273.15; Tc=508.1; Pc=4.6924; vL=7.145e-5; vV=0.2453;
>> dHv = vh Wagner(T,Tc,Pc,vL,vV,C)
dHv =
3.3015e+04

```

The result is $\Delta H_v = 33,015$ J/mol. The experimental value is reported to be 32,460.9 J/mol.²⁶ We can see that the deviation is about 1.7%.

3.10 HEAT OF FORMATION FOR IDEAL GASES

The correlation of the heat of formation, ΔH_f^0 , of the ideal gas at low temperature can be expressed as²⁷

$$\Delta H_f^0 = A + BT + CT^2$$

where

ΔH_f^0 is the heat of formation of ideal gas at low temperature (kcal/gmol)

T is the temperature (K)

A , B , and C are correlation constants

The MATLAB function *hform* calculates the heat of formation of gases listed in Table 3.3. This function defines constants A , B , and C in the range of 298 K $\leq T \leq$ 1500 K. The basic syntax is

```
hf = hform(T, cname)
```

where T is the temperature (°C); $cname$ is the name of the component, which can be specified as a chemical formula or common name; and hf is the estimated heat of formation (kcal/gmol) at T .

```

function hf = hform(T,cname)
% Estimation of heat of formation of gases (kcal/gmol) at low temperature
% input
% T: temperature(C) (scalar or row vector)
% cname: name or chemical formula of the compound
% output
% hf: heat of formation (kcal/gmol)

% correlation constants (A, B, C)
cv = [0.0 0.0 0.0 % missing component(F2)
      0.0 0.0 0.0 % missing component(Cl2)
      -69.6000 -5.2900 0
      -26.5000 0.8400 -1.0500
      -93.9000 -0.4300 0
      -21.9000 -0.6100 0
      -9.3400 -6.2000 2.3600
      -57.4000 -1.7900 0
      -31.8000 -3.0400 1.1900
      0.0 0.0 0.0 % missing component(H2)
      0.0 0.0 0.0 % missing component(N2)
      0.0 0.0 0.0 % missing component(O2)
      0.0 0.0 0.0 % missing component(C2H4)
      -15.4000 -9.5900 3.5000
      -16.4000 -14.8000 6.1300
      -20.0000 -19.1000 8.1500
      23.7000 -15.3000 6.2700
      16.8000 -19.0000 7.8400
      25.2000 -17.6000 8.9800
      -19.3000 -14.6000 7.1800
      16.9000 -16.7000 7.9000
      -21.6000 -32.0000 15.8000
      29.0000 -10.1000 4.1300
      -44.9600 -11.9000 4.9800
      -24.7000 0.0334 0
      -24.0000 2.4200 0];
wv = [cv(:,1) cv(:,2)*1e-3 cv(:,3)*1e-6];
ind = compID(cname); % identification number of the component
% computes heat of formation
T0 = 273.15; T = T + T0;
if ind ~= 3
    hf = wv(ind,1) + wv(ind,2)*T + wv(ind,3)*T.^2;
else % ind = 3: sulfur dioxide
    hf = [];
    for j = 1:length(T)
        if T(j) >= 298 && T(j) <= 717 % temperature range: 298K~717K
            ind = 1;
            hf = wv(ind,1) + wv(ind,2)*T(j) + wv(ind,3)*T(j).^2;
        else % temperature range: 717K~1500K
            wv(ind,:) = [-86.9 0.32*1e-3 0];
            hf = wv(ind,1) + wv(ind,2)*T(j) + wv(ind,3)*T(j).^2;
        end
        hf = [hf hf];
    end
end
end

```

Example 3.19: Standard Heat of Formation of Methane

Estimate the heat of formation of methane (kcal/gmol) at 500°C.

Solution

```
>> T = 500;
>> hf = hform(T, 'methane')
hf =
-20.7223
```

3.11 GIBBS FREE ENERGY

The Gibbs free energy of formation, ΔG_f^0 , is defined as:

$$\Delta G_f^0 = \Delta H_f^0 - T \Delta S_f^0$$

where ΔS_f^0 is the entropy of formation. The Gibbs free energy of formation of the ideal gas at low pressure can be expressed as a linear relationship in temperature as²⁸

$$\Delta G_f^0 = A + BT$$

where

ΔG_f^0 is the Gibbs free energy of formation of ideal gas at low pressure (kcal/gmol)

T is the temperature (K)

A and B are correlation constants

The MATLAB function *gfree* calculates the Gibbs free energy of formation of components listed in [Table 3.3](#). This function defines constants A and B. This function is used as

```
gf = gfree(T, cname)
```

where T is the temperature (K); cname is the name of the component, which can be specified as a chemical formula or common name; and gf is the estimated Gibbs free energy of formation (kcal/gmol) at T. T can be either a scalar or a row vector representing a temperature range.

```
function gf = gfree(T, cname)
% Estimation of Gibbs free energy of gases (kcal/gmol) at low pressure
% input
% T: temperature(C) (scalar or row vector)
% cname: name or chemical formula of the compound
% output
% gf: Gibbs free energy of formation (kcal/gmol)

% correlation constants (A, B)
cv = [ 0.0 0.0 % missing component(F2)
        0.0 0.0 % missing component(Cl2)
        -71.9000 0.2500
        -26.5000 -21.3000
        -94.2000 -0.4200
        -22.3000 -1.7200
        -12.3000 27.2000
        -58.6000 12.7000
        -33.2000 26.4000
```

```

0.0 0.0 % missing component (H2)
0.0 0.0 % missing component (N2)
0.0 0.0 % missing component (O2)
0.0 0.0 % missing component (C2H4)
-20.1000 24.9000
-23.3000 49.7000
-28.8000 74.7000
16.7000 45.9000
7.8000 68.7000
18.5000 70.6000
-25.0000 56.5000
10.3000 47.7000
-35.1000 139.0000
24.2000 38.6000
-50.2000 36.5000
-24.8000 26.9000
-22.6000 32.1000] ;

wv = [cv(:,1) cv(:,2)*1e-3];
ind = compID(cname); % identification number of the component
% computes Gibbs free energy
if ind ~= 3
    gf = wv(ind,1) + wv(ind,2)*T;
else % ind = 3: sulfur dioxide
    gf = [];
    for j = 1:length(T)
        if T(j) >= 298 && T(j) <= 717 % temperature range: 298K~717K
            ind = 1;
            gfv = wv(ind,1) + wv(ind,2)*T(j);
        else % temperature range: 717K~1500K
            wv(ind,:) = [-86.8 17.7*1e-3];
            gfv = wv(ind,1) + wv(ind,2)*T(j);
        end
        gf = [gf gfv];
    end
end
end

```

For example, the Gibbs free energy of formation of benzene at 60°C can be obtained as

```

>> gfree(60+273.15, 'Benzene')
ans =
    31.9916

```

3.12 DIFFUSION COEFFICIENTS

3.12.1 LIQUID-PHASE DIFFUSION COEFFICIENTS

Diffusion coefficients in liquid phase depend on the concentration and are valid for dilute solutions with solute concentrations less than 10 mol%. For a binary mixture of solute *A* dissolved in solvent *B*, the diffusion coefficient can be represented as D_{AB}^0 for concentrations of *A* up to 10 mol%. The Wilke–Chang method can be used to estimate D_{AB}^0 ²⁹:

$$D_{AB}^0 = \frac{7.4 \cdot 10^{-8} T \sqrt{\phi M_{wB}}}{\mu_B \cdot V_A^{0.6}}$$

where

D_{AB}^0 is the diffusion coefficient of solute A at very low concentration in B (cm²/s)

M_{wB} is the molecular weight of solvent B

T is the temperature (K)

μ_B is the viscosity of solvent B (cP)

V_A is the molal volume of solute A at its normal boiling point (cm³/gmol)

ϕ is the dimensionless association factor of solvent B: 2.6 for water, 1.9 for methanol, 1.5 for ethanol, and 1.0 for unassociated solvents

The MATLAB function *gdiffc* estimates diffusion coefficients in liquid phase. The basic syntax of this function is

```
Df = gdiffc(T,M,phi,mu,v)
```

where T is the temperature (K) (a scalar or a vector), M is the molecular weight, phi is the association factor of solvent B, mu is the viscosity of solvent B (cP), v is the molal volume of solute A at its normal boiling point (cm³/gmol), and Df is the estimated diffusion coefficient in liquid phase (cm²/s).

```
function Df = gdiffc(T,M,phi,mu,v)
% Estimation of diffusion coefficients in liquid phase
% input
% T: temperature(C) (scalar or row vector)
% M: molecular weight
% phi: association factor of solvent B
% mu: viscosity of solvent B (cP)
% v: molal volume of solute A at its normal boiling point (cm^3/gmol)
% output
% Df: estimated diffusion coefficient in liquid phase (cm^2/s)

Df = 7.4e-8*T.* sqrt(phi.*M)./(mu.*v.^0.6);
end
```

3.12.2 GAS-PHASE DIFFUSION COEFFICIENTS

Diffusion coefficients for nonpolar gases can be estimated from Fuller, Schettler, and Giddings's method, which is expressed as³⁰

$$D_{AB}^0 = \frac{10^{-3} T^{1.75} \sqrt{(M_{wA} + M_{wB}) / (M_{wA} M_{wB})}}{P \left[\left(\sum v \right)_A^{1/3} + \left(\sum v \right)_B^{1/3} \right]^2}$$

where

D_{AB}^0 is the diffusion coefficient for a binary mixture of gases A and B (cm²/s)

M_{wA} and M_{wB} are molecular weights of A and B, respectively

v_A and v_B are atomic diffusion volumes of A and B, respectively

T is temperature (K)

P is pressure (atm)

The summation of the diffusion volume coefficients for components A and B, $\sum v$, is shown in Table 3.5.

TABLE 3.5
Atomic Diffusion Volumes for Use in Fuller, Schettler, and Giddings's Method

Atomic and Structural Diffusion Volume Increments, v

C	16.5	Cl	19.5
H	1.98	S	17.0
O	5.48	Aromatic ring	-20.2
N	5.69	Heterocyclic ring	-20.2

Diffusion Volumes for Simple Molecules, $\sum v$

H ₂	7.07	CO	18.9
D ₂	6.70	CO ₂	26.9
He	2.88	N ₂ O	35.9
N ₂	17.9	NH ₃	14.9
O ₂	16.6	H ₂ O	12.7
Air	20.1	CCl ₂ F ₂	114.8
Ar	16.1	SF ₆	69.7
Kr	22.8	Cl ₂	37.7
Xe	37.9	Br ₂	67.2
SO ₂	41.1		

Source: Coker, A.K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, 1995, p. 124.

3.13 COMPRESSIBILITY FACTOR OF NATURAL GASES

The compressibility factor Z of natural hydrocarbon gases can be estimated by using the revised Awoseyin method³²:

$$Z = F_1 \left[\frac{1}{1 + \left(\left(A_6 \cdot P \cdot 10^{1.785 S_g} \right) / T^{3.825} \right)} + F_2 \cdot F_3 \right] + F_4 + F_5$$

where

T is temperature (R)

P is pressure (Kpsia)

$$F_1 = P (0.251 S_g - 0.15) - 0.202 S_g + 1.106$$

$$F_2 = 1.4 e^{-0.0054(T-460)}$$

$$F_3 = A_1 P^5 + A_2 P^4 + A_3 P^3 + A_4 P^2 + A_5 P$$

$$F_4 = (0.154 - 0.152 S_g) P^{(3.18 S_g - 1.0)} e^{-0.5 P} - 0.02$$

$$F_5 = 0.35 (0.6 - S_g) e^{-1.039(P-1.8)^2}$$

and the values of the constants are

$$A_1 = 0.001946, \quad A_2 = -0.027635, \quad A_3 = 0.136315, \quad A_4 = -0.23849$$

$$A_5 = 0.105168, \quad A_6 = 3.44 \times 10^8$$

The specific gravity, S_g , of natural gas can be calculated from its density or molecular weight:

$$S_g = \frac{(\text{Density of gas})}{(\text{Density of air})} = \frac{\rho_{\text{gas}, 60^\circ\text{F}}}{\rho_{\text{air}, 60^\circ\text{F}}} \quad \text{or} \quad S_g = \frac{\text{Molecular weight of gas}}{\text{Molecular weight of air}} = \frac{M_{w,\text{gas}}}{M_{w,\text{air}}}$$

The MATLAB function *ngasZ* estimates the compressibility factor Z of natural hydrocarbon gases. This function can be used as

```
nz = ngasZ(T, P, Sg)
```

where T is temperature ($^\circ\text{F}$) (a scalar or a vector), P is pressure (psia) (a scalar or a vector), Sg is the specific gravity of the natural gas, and nz is the estimated compressibility factor of the natural gas at T and P. The units of T and P are automatically converted to R and Kpsia, respectively, within the function.

```
function nz = ngasZ(T, P, Sg)
% Estimates the compressibility factor Z of natural gases
% input
% T: temperature (F) (scalar or vector)
% P: pressure (psia)
% Sg: specific gravity of the natural gas
% output
% nz: estimated compressibility factor Z of natural gases

P = P/1000; T = T + 460;
A1 = 0.001946; A2 = -0.027635; A3 = 0.136315;
A4 = -0.23849; A5 = 0.105168; A6 = 3.44e8;
F1 = P.* (0.251*Sg-0.15) - 0.202*Sg + 1.106;
den = 1 + A6.*P.*10.^(1.785*Sg)./(T.^3.825);
F2 = 1.4.*exp(-0.0054*(T-460));
F3 = A1.*P.^5 + A2.*P.^4 + A3.*P.^3 + A4.*P.^2 + A5.*P;
F4 = (0.154-0.152*Sg).*P.^ (3.18*Sg-1).*exp(-0.5*P) - 0.02;
F5 = 0.35*(0.6-Sg).*exp(-1.039*(P-1.8).^2);
nz = F1.* (1./den + F2.*F3) + F4 + F5;
end
```

Example 3.20: Compressibility Factor of Natural Gases³³

Estimate compressibility factors of natural gases at 60°F and at the pressure range of $100 \leq P \leq 5000$ (psia) when the specific gravity is 0.5, 0.6, 0.7, and 0.8. Plot the results as a function of pressure and specific gravity.

Solution

The following shows the calculation procedure and produces the curves shown in [Figure 3.6](#).

```
>> T=60; P=100:10:5000; nz=[];
>> for k = 1:4, Sg = 0.5+(k-1)*0.1; nzv = ngasZ(T,P,Sg); nz = [nz nzv']; end
>> plot(P,nz(:,1),P,nz(:,2),':',P,nz(:,3),'.-',P,nz(:,4),'- -')
```

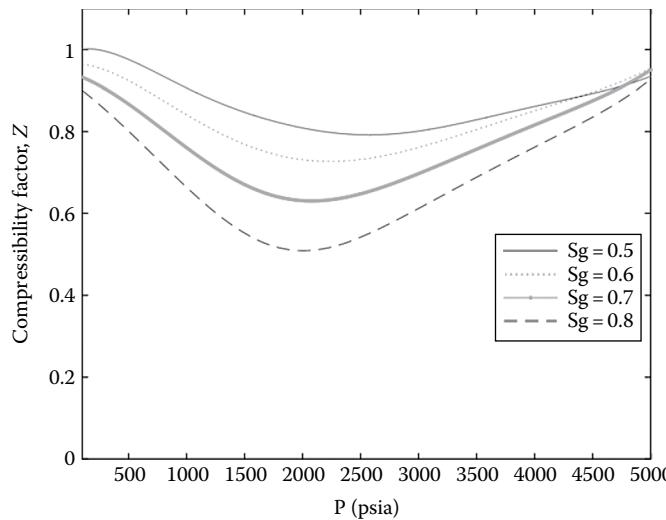


FIGURE 3.6 Plot of compressibility factors of natural gases as a function of pressure and specific gravity.

```
>> axis([100 5000 0 1.1]), legend('Sg=0.5','Sg=0.6','Sg=0.7','Sg=0.8','location','best')
>> xlabel('P(psia)'), ylabel('Compressibility factor, Z')
```

PROBLEMS

- 3.1 Estimate the physical properties of saturated steam for $200 \leq T \leq 300$ ($^{\circ}\text{C}$) at 20°C intervals and tabulate the results.
- 3.2 Calculate the values of absolute humidity (g/m^3) when the dry bulb temperature (T_d) increases from 10°C to 40°C at 10°C intervals and the relative humidity varies from 20% to 80% at 20% intervals.
- 3.3 In a storage room, the initial air temperature is 28°C and the relative humidity is 73%. After a while, the air temperature and the relative humidity decreased to 16°C and 55%, respectively. Assuming that the door of the room was kept closed, calculate the amount of water condensed. The volume of the air in the room is 150 m^3 .
- 3.4 Calculate the density of water for $0 \leq T \leq 350$ ($^{\circ}\text{C}$) at 1°C intervals and plot the results as a function of temperature.
- 3.5 Calculate the viscosity of water for $0 \leq T \leq 350$ ($^{\circ}\text{C}$) and plot the results as a function of temperature.
- 3.6 Estimate the viscosity of ethane for $25 \leq T \leq 900$ ($^{\circ}\text{C}$) at 1°C intervals and plot the results as a function of temperature.
- 3.7 Estimate the heat capacity of water for $0 \leq T \leq 350$ ($^{\circ}\text{C}$) and plot the results as a function of temperature.
- 3.8 Estimate the heat capacity of carbon dioxide for $300 \leq T \leq 1500$ (K) at 1 K intervals and plot the results as a function of temperature.
- 3.9 Estimate the thermal conductivity of phenol for $0 \leq T \leq 350$ ($^{\circ}\text{C}$) at 1°C intervals and plot the results as a function of temperature.
- 3.10 Estimate the thermal conductivities of carbon dioxide and methane for $20 \leq T \leq 500$ ($^{\circ}\text{C}$) and plot the results as a function of temperature.
- 3.11 Calculate the surface tension of water (dynes/cm) for $0 \leq T \leq 350$ ($^{\circ}\text{C}$) and plot the results as a function of temperature.

TABLE P3.13
Each Group and Frequency of the RE-218 Molecule

Group	v (Frequency)	ΔB_i
C	3	-0.0896
F_3	2	0.09402
F_2	1	0.1054
O	1	0.15049
No H	1	-0.19373

- 3.12** Calculate the vapor pressures of benzene and phenol (mmHg) for $0 \leq T \leq 300$ (°C) and plot the results as a function of temperature.
- 3.13** Estimate the vapor pressure of RE-218 ($CF_3-O-CF_2-CF_3$) at 50.3°C and at -45°C by using the Rarey/Moller method. The normal boiling point is $T_b = -23.7$ °C. Compare the results with experimental values (11.319 bar at 50.3°C and 0.372 bar at -45°C).³⁴ Each group and frequency of the RE-218 molecule are shown in Table P3.13.
- 3.14** Estimate the enthalpy of vaporization of acetone at 273.15 K by using the derivative relation based on the Antoine equation. The parameters of the Antoine equation are $A = 7.11714$, $B = 1210.595$, and $C = 229.664$.
- 3.15** Estimate the standard heat of formation of CO and CO_2 for $25 \leq T \leq 1200$ (°C) and plot the results as a function of temperature.
- 3.16** Calculate the Gibbs free energy of formation (kcal/gmol) of ethane for $300 \leq T \leq 1500$ (K) and plot the results as a function of temperature.
- 3.17** Determine the value of the infinite-dilution diffusion coefficient of propane (A) in chlorobenzene (B) at 0°C by using the Wilke–Chang relation. The molecular weight of chlorobenzene is 112.56, $\mu = 1.05$ cP, and the molar volume of propane is $74.5 \text{ cm}^3/\text{gmol}$, $T = 273.15$ K, and $\varphi = 1.0$.
- 3.18** Estimate the diffusion coefficient of allyl chloride in air at 25°C and 1 atm. Compare the result with the experimental value of $0.0975 \text{ cm}^2/\text{s}$.³⁵

REFERENCES

1. International Association for the Properties of Water and Steam, *Revised Release of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam*, IAPWS, Lucerne, Switzerland, August 2007.
2. International Association for the Properties of Water and Steam, *Revised Release on the IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use*, IAPWS, Lucerne, Switzerland, September 2016.
3. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, p. 719, 2005.
4. Yaws, C. L. et al., *Physical Properties*, A Chemical Engineering Publication, McGraw-Hill, New York, NY, 1977.
5. Gmehling, J., B. Kolbe, M. Kleiber, and J. Rarey, *Chemical Thermodynamics for Process Simulation*, Wiley-VCH Verlag GmbH & Co., Weinheim, Germany, pp. 95–96, 2012.
6. Gmehling, J., B. Kolbe, M. Kleiber, and J. Rarey, *Chemical Thermodynamics for Process Simulation*, Wiley-VCH Verlag GmbH & Co., Weinheim, Germany, p. 96, 2012.
7. Andrade, E. N. d. C., Properties of dense gases and liquids, *Philosophical Magazine*, 17, 497, 698, 1934.
8. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 121, 1995.
9. Poling, B. E., J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, pp. 9–21, 2001.

10. Poling, B. E., J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, pp. 9–22, 2001.
11. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 109, 1995.
12. Gmehling, J., B. Kolbe, M. Kleiber, and J. Rarey, *Chemical Thermodynamics for Process Simulation*, Wiley-VCH Verlag GmbH & Co., Weinheim, Germany, p. 111, 2012.
13. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 110, 1995.
14. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 111, 1995.
15. Poling, B. E., J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, pp. 12–13, 12–18, 2001.
16. Poling, B. E., J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, pp. 12–18, 2001.
17. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 112, 1995.
18. Poling, B. E., J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, p. 7, 2001.
19. Gmehling, J., B. Kolbe, M. Kleiber, and J. Rarey, *Chemical Thermodynamics for Process Simulation*, Wiley-VCH Verlag GmbH & Co., pp. 84–85, 2012.
20. Gmehling, J., B. Kolbe, M. Kleiber, and J. Rarey, *Chemical Thermodynamics for Process Simulation*, Wiley-VCH Verlag GmbH & Co., p. 86, 2012.
21. Gmehling, J., B. Kolbe, M. Kleiber, and J. Rarey, *Chemical Thermodynamics for Process Simulation*, Wiley-VCH Verlag GmbH & Co., p. 89, 2012.
22. Gmehling, J., B. Kolbe, M. Kleiber, and J. Rarey, *Chemical Thermodynamics for Process Simulation*, Wiley-VCH Verlag GmbH & Co., pp. 89–90, 2012.
23. Poling, B. E., J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, pp. 7–24, 2001.
24. Pitzer, K. S., D. Lippman, R. F. Curl, C. M. Higgins, and D. E. Peterson, A new correlation method for enthalpies of vaporization, *Journal of the American Chemical Society*, 77, 3433, 1955.
25. Gmehling, J., B. Kolbe, M. Kleiber, and J. Rarey, *Chemical Thermodynamics for Process Simulation*, Wiley-VCH Verlag GmbH & Co., Weinheim, Germany, pp. 101–102, 2012.
26. Gmehling, J., B. Kolbe, M. Kleiber, and J. Rarey, *Chemical Thermodynamics for Process Simulation*, Wiley-VCH Verlag GmbH & Co., Weinheim, Germany, pp. 102–103, 2012.
27. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 115, 1995.
28. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, pp. 118–119, 1995.
29. Poling, B. E., J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, pp. 11.21–11.22, 2001.
30. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 123, 1995.
31. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 124, 1995.
32. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 125, 1995.
33. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, pp. 126–127, 1995.
34. Gmehling, J., B. Kolbe, M. Kleiber, and J. Rarey, *Chemical Thermodynamics for Process Simulation*, Wiley-VCH Verlag GmbH & Co., Weinheim, Germany, p. 91, 2012.
35. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 127, 1995.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

4 Thermodynamics

In chemical process modeling and simulation, the degree of accuracy of thermodynamic properties, phase equilibria, mass and heat transfer, and chemical reactions mainly determines the quality of the model and simulation. Since these parameters are strongly influenced by thermodynamic relationships, a proper application of chemical engineering thermodynamic principles is an essential requirement of a successful process modeling and simulation. Chemical engineering thermodynamics deals with real substances whose properties are not entirely known from experiment at all possible temperatures and pressures and therefore are approximations described by model equations. These equations include volumetric equations of state that interrelate pressure, volume, and temperature, and equations that relate activity coefficients to compositions. Practicing engineers often need to solve problems even when descriptions of physical properties are imperfect, and a selection of equation of state or activity coefficient model must be made. Typically, the equations of state and activity coefficient models used in chemical engineering thermodynamics are not simple linear algebraic equations, so computations involving them may be difficult.

The main objective of this chapter is to provide readers with various thermodynamic models and corresponding MATLAB® programs that have already been found—or potentially would become—useful in academic and industrial applications. The MATLAB programs in this chapter can be used in undergraduate or graduate courses on chemical engineering thermodynamics, provided that students have a prerequisite understanding of the fundamentals of chemical engineering. Researchers and practicing engineers in the field of chemical engineering can use these MATLAB programs in the modeling and simulation of chemical processes.

4.1 EQUATION OF STATE

4.1.1 VIRIAL STATE EQUATION

The virial equation of state is expressed as¹

$$Z = \frac{PV}{RT} = 1 + \frac{BP}{RT}$$

where

P is pressure (atm)

V is molar volume (liter/g mol)

T is temperature (K)

R is gas constant (0.08206 atm · liter/(g mol · K))

B is a second virial coefficient (liter/g mol) given by²

$$B = \frac{RT_c}{P_c} (B^0 + wB^1), \quad B^0 = 0.083 - \frac{0.422}{T_r^{1.6}}, \quad B^1 = 0.139 - \frac{0.172}{T_r^{4.2}}$$

where

T_c is the critical temperature (K)

P_c is the critical pressure (atm)

w is acentric factor

T_r and P_r are reduced temperature and pressure defined as

$$T_r = \frac{T}{T_c}, \quad P_r = \frac{P}{P_c}.$$

The MATLAB function *virialeOS* evaluates the virial equation of state. This function computes the compressibility factor and molar volume (liter/g mol) at a given temperature (K) and pressure (atm). A simple expression of its syntax is

```
[Z V] = virialeOS(P,T,Pc,Tc,w)
```

Here, P and Pc are the pressure and critical pressure (atm), respectively; T and Tc are the temperature and critical temperature (K), respectively; w is the acentric factor; and Z and V are the compressibility factor and molar volume, respectively.

```
function [Z V] = virialeOS(P,T,Pc,Tc,w)
% Estimation of compressibility factor and molar volume
% at given T and P using the virial equation of state
% inputs:
%     P,Pc: pressure and critical pressure (atm)
%     T,Tc: temperature and critical temperature (K)
%     w: acentric factor
% outputs:
%     Z: compressibility factor
%     V: molar volume

R = 0.08206; % atm-liter/(gmol-K)
Tr = T./Tc; Pr = P./Pc;
B0 = 0.083 - 0.422./ (Tr.^1.6); B1 = 0.139 - 0.172./ (Tr.^4.2);
B = R*Tc.* (B0 + w.*B1)./Pc;
Z = 1 + B.*P./ (R*T); V = R*Z.*T./P;
end
```

Example 4.1: Compressibility Factor and Molar Volume of Ethane³

Use the virial equation of state to determine the compressibility factor and the molar volume of ethane at 50°C and 15 bar. For ethane, $T_c = 305.3$ K, $P_c = 48.08$ atm, and $w = 0.1$.

Solution

Since 1 bar = 0.986923 atm, $P = (15)(0.986923) = 14.8$ atm. The following commands produce the compressibility factor and the molar volume of ethane:

```
>> P=14.8; T=323.15; Pc=48.08; Tc=305.3; w=0.1;
>> [Z V] = virialeOS(P,T,Pc,Tc,w)
Z =
    0.9122
V =
    1.6344
```

4.1.2 LEE-KESLER EQUATION

The state equations in which the compressibility factor Z is represented as a function of T_r and P_r can be applied to most gases and are said to be generalized. Lee and Kesler⁴ generalized the BWR (Benedict–Webb–Rubin) equation using the corresponding state principle to predict the properties of a variety of compounds including nonhydrocarbons. They extended the Curl–Pitzer method, which is based on the corresponding state principle to lower temperature range, and expressed the compressibility factor of a liquid in terms of the compressibility factor of the simple fluid ($\omega = 0$), $Z^{(0)}$, and that of a reference fluid (*n*-octane), $Z^{(r)}$. They employed the revised BWR (Benedict–Webb–Rubin) equation to express $Z^{(0)}$ and $Z^{(r)}$:

$$Z = Z^{(0)} + \frac{w}{w^{(r)}} (Z^{(r)} - Z^{(0)})$$

$$Z = \left(\frac{P_r V_r}{T_r} \right) = 1 + \frac{B}{V_r} + \frac{C}{V_r^2} + \frac{D}{V_r^5} + \frac{c_4}{T_r^3 V_r^2} \left(\beta + \frac{\gamma}{V_r^2} \right) \exp \left(-\frac{\gamma}{V_r^2} \right)$$

where

$$B = b_1 - \frac{b_2}{T_r} - \frac{b_3}{T_r^2} - \frac{b_4}{T_r^3}, \quad C = c_1 - \frac{c_2}{T_r} + \frac{c_3}{T_r^3}, \quad D = d_1 + \frac{d_2}{T_r}$$

The constants b_i , c_i ($i = 1, 2, 3, 4$), d_1 , d_2 , β , and γ are shown in Table 4.1.

The MATLAB function *zLK* calculates the compressibility factor Z using the Lee–Kesler equation. This function can be called as

$Z = zLK(g)$

The required data including temperature and pressure are supplied as a structure variable *g*. The structure *g* contains several fields including pressure (*g.P*, MPa), temperature (*g.T*, K), critical pressure (*g.Pc*, MPa), critical temperature (*g.Tc*, K), and acentric factor (*g.w*).

TABLE 4.1
Constants of the Lee–Kesler Equation

Constants	Simple Fluid	Reference Fluid
b_1	0.1181193	0.2026579
b_2	0.265728	0.331511
b_3	0.154790	0.027665
b_4	0.030323	0.203488
c_1	0.0236744	0.0313385
c_2	0.0186984	0.0503618
c_3	0.0	0.041577
c_4	0.042724	0.041577
$10^4 d_1$	0.155488	0.48736
$10^4 d_2$	0.623689	0.0740336
β	0.65392	1.226
γ	0.060167	0.03754

Source: Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, 1985, p. 23.

```

function Z = zLK(g)
% zLK.m: calculates compressibility factor using the Lee-Kesler equation
% constants and parameters
b = [0.1181193 0.2657280 0.1547900 0.0303230;
      0.2026579 0.3315110 0.0276550 0.2034880];
c = [0.0236744 0.0186984 0.0000000 0.0427240;
      0.0313385 0.0503618 0.0169010 0.0415770];
d = 1e-4*[0.155488 0.623689; 0.487360 0.0740336];
beta = [0.653920 1.22600]; gam = [0.060167 0.03754]; wr = 0.3978;
% Assign P(MPa), T(K), Pc(MPa), Tc(K), w(acentric factor)
P = g.P; T = g.T; Pc = g.Pc; Tc = g.Tc; w = g.w;
Pr = P/Pc; Tr = T/Tc;
h.c = c; h.Tr = Tr; h.Pr = Pr; h.gam = gam; h.beta = beta;
% Calculation of Z0 of simple fluid (ind=1)
ind=1; [B C D] = compBCD(Tr, b, c, d, ind);
h.B = B; h.C = C; h.D = D; h.ind = ind; Vr0 = Tr./Pr; % initial Vr
Vr = fzero(@BWReq, Vr0, [], h); Z0 = Pr*Vr./Tr;
% Calculation of Zr of reference fluid (ind=2)
ind = 2; [B C D] = compBCD(Tr, b, c, d, ind);
h.B = B; h.C = C; h.D = D; h.ind = ind;
Vr = fzero(@BWReq, Vr, [], h); Zr = Pr*Vr/Tr;
Z1 = 1/wr * (Zr - Z0);
% compressibility factor of real fluid
Z = Z0 + w*Z1;
fprintf('Compressibility factor Z = %f\n', Z);
end

```

The function *zLK* first computes the constants of the Lee–Kesler equation B, C, and D using the function *compBCD* followed by the determination of the roots of the BWR equation. The BWR equation is defined by the function *BWReq* where the required data is supplied by the structure variable *h*.

```

function [B C D] = compBCD(Tr, b, c, d, ind)
% ind 1: Simple fluids, ind 2: Reference fluids
B = b(ind,1) - b(ind,2)/Tr - b(ind,3)/Tr^2 - b(ind,4)/Tr^3;
C = c(ind,1) - c(ind,2)/Tr + c(ind,3)/Tr^3;
D = d(ind,1) + d(ind,2)/Tr;
end

function f = BWReq(Vr,h)
B = h.B; C = h.C; D = h.D; c = h.c; Tr = h.Tr; Pr = h.Pr; gam = h.gam;
beta = h.beta; ind = h.ind;
f = 1 + B/Vr + C/Vr^2 + D/Vr^5 + c(ind, 4)/(Tr^3*Vr^2)*(beta(ind)+...
          gam(ind)/Vr^2)*exp(-gam(ind)/Vr^2) - Pr*Vr/Tr;
end

```

Example 4.2: Compressibility Factor of 1-Butene

Determine the compressibility factor of 1-butene at 400 K and 20 MPa. For 1-butene, the critical temperature is 419.6 K, the critical pressure is 40.2 MPa, and the acentric factor is 0.191.

Solution

Values are assigned first to corresponding fields of the structure variable *g*. The function *zLK* produces the compressibility factor of 1-butene at given conditions.

```

>> g.P=20; g.T=400; g.Pc=40.2; g.Tc=419.6; g.w=0.191;
>> Z = zLK(g);
Compressibility factor Z = 0.753883

```

4.1.3 CUBIC EQUATIONS OF STATE

For vapor, the cubic equation of state is written as

$$V = \frac{RT}{P} + b - \frac{a(T)}{P} \frac{V-b}{(V+\epsilon b)(V+\sigma b)}$$

and for liquid, it is written as

$$V = b + (V+\epsilon b)(V+\sigma b) \left[\frac{RT + bP - VP}{a(T)} \right]$$

These equations can be rearranged as a 3rd-order polynomial equation with respect to V , thus the cubic equation. Here, a and b are given by

$$a(T) = \Psi \frac{\alpha(T_r) R^2 T_c^2}{P_c}, \quad b = \Omega \frac{RT_c}{P_c}$$

Equations for Z equivalent to above equations can be obtained by substituting the compressibility factor $Z = PV/RT$. From the vapor equation, we have

$$Z = 1 + \beta - q\beta \frac{Z - \beta}{(Z + \epsilon\beta)(Z + \sigma\beta)}$$

and for liquid,

$$Z = \beta + (Z + \epsilon\beta)(Z + \sigma\beta) \left(\frac{1 + \beta - Z}{q\beta} \right)$$

where $\beta = bP/RT = \Omega(P_c/T_r)$, $q = (a(T))/bRT = (\Psi\alpha(T_r))/\Omega T_r$. These equations can be rearranged to give a 3rd-order polynomial equation:

$$Z^3 + \{(\sigma + \epsilon)\beta - (1 + \beta)\}Z^2 + \beta\{q + \epsilon\sigma\beta - (1 + \beta)(\sigma + \epsilon)\}Z - \beta^2\{q + (1 + \beta)\epsilon\sigma\} = 0$$

Parameters for each state equation type are given in [Table 4.2](#).

The MATLAB function *cubicEOSZ* implements the cubic equation of state. This function uses the built-in function *fzero* to estimate the compressibility factor and molar volume. A simple calling syntax is

```
[Z V] = cubicEOSZ(state, eos, T, P, Tc, Pc, w)
```

Here, *state* denotes the state of the fluid: 'l' or 'L' when the fluid is liquid, 'v' or 'V' when the fluid is vapor. *eos* is the equation of state being used: 'VDW' when the van der Waals equation is used, 'RK' when the Redlich–Kwong equation is used, 'SRK' when the Soave–Redlich–Kwong equation is used, and 'PR' when the Peng–Robinson equation is used (both capital and lowercase letters are permitted). *T* and *P* are the temperature (K) and pressure (bar), respectively; *Tc* and *Pc* are the critical temperature (K) and pressure (bar), respectively; and *w* is the acentric factor. *Z* and *V* are estimated values of the compressibility factor and the molar volume (cm³/mol).

TABLE 4.2
Parameters for Equations of State

Equation of State	$\alpha(T_r)$	σ	ϵ	Ω	Ψ
van der Waals	1	0	0	0.12500	0.42188
Redlich–Kwong	$T_r^{-1/2}$	1	0	0.08664	0.42748
Soave–Redlich–Kwong	α_{SRK}	1	0	0.08664	0.42748
Peng–Robinson	α_{PR}	$1 + \sqrt{2}$	$1 - \sqrt{2}$	0.07780	0.45724
	$\alpha_{SRK}(T_r) = \left[1 + \left(0.480 + 1.574w - 0.176w^2 \right) \left(1 - \sqrt{T_r} \right) \right]^2$				
	$\alpha_{PR}(T_r) = \left[1 + \left(0.37464 + 1.54226w - 0.26992w^2 \right) \left(1 - \sqrt{T_r} \right) \right]^2$				

Source: Smith, J.M. et al., *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, 2005, p. 98.

```

function [Z V] = cubicEOSZ(state,eos,T,P,Tc,Pc,w)
% Estimation of Z and V using cubic equations of state
% input
%     state: fluid state (liquid: L, vapor: V)
%     eos: type of equation being used (VDW, RK, SRK, PR)
%     T,P: temperature (K) and pressure (bar)
%     Tc,Pc: critical temperature (K) and pressure (bar)
%     w: acentric factor

% Tr and Pr (reduced T and P)
Tr = T/Tc; Pr = P/Pc; R = 83.14; % cm^3*bar/mol/K
nc = length(w);

% set parameters
eos = upper(eos);
switch eos
    case {'VDW'}
        sm = 0; ep = 0; om = 0.125; ps = 0.42188; mx = [0];
    case{'RK'}
        ep = 0; sm = 1; om = 0.08664; ps = 0.42748;
        al = 1./sqrt(Tr);
        mx = (Tr.^(-1/4) - 1)./(1 - sqrt(Tr));
    case{'SRK'}
        ep = 0; sm = 1; om = 0.08664; ps = 0.42748;
        al = (1+(0.48+1.574*w-0.176*w.^2).* (1-sqrt(Tr))).^2;
        mc = [0.48 1.574 0.176];
        mx = [ones(nc,1) w -w.^2]*mc';
    case{'PR'}
        ep = 1 - sqrt(2); sm = 1 + sqrt(2); om = 0.07780; ps = 0.45724;
        mc = [0.37464 1.54226 0.26992];
        mx = [ones(nc,1) w -w.^2]*mc';
    end

% calculation of alpha, beta and q
al = (1 + mx.* (1-sqrt(Tr))).^2;
beta = om*Pr./Tr; q = ps*al./ (om*Tr);

```

```
% calculation of Z and V
state = upper(state);
c(1) = 1;
c(2) = (sm +ep)*beta - (1+beta);
c(3) = beta*(q + ep*sm*beta -(1+beta)*(sm+ep));
c(4) = -beta^2*(q +(1+beta)*ep*sm);

Z = roots(c);
switch state
    case 'V'
        Z = max(Z);
    case 'L'
        Z = min(Z);
end
V = Z*R*T/P;
end
```

Example 4.3: Molar Volume of *n*-Butane⁷

Determine the molar volumes of saturated vapor and saturated liquid *n*-butane using the cubic equations of state (the van der Waals equation, the Redlich–Kwong equation, the Soave–Redlich–Kwong equation, and the Peng–Robinson equation). For *n*-butane, the vapor pressure at 350 K is 9.4573 bar, $T_c = 425.1$ K, $P_c = 37.96$ bar, and $w = 0.2$.

Solution

The following commands show the calculation procedure for the saturated vapor using the van der Waals equation:

```
>> T=350; P=9.4573; Tc=425.1; Pc=37.96; w=0.2;
>> state = 'v'; eos = 'vdw';
>> [Z V] = cubicEOSZ(state,eos,T,P,Tc,Pc,w)
Z =
    0.8667
V =
    2.6669e+03
```

The calculation procedure for the saturated liquid using the Redlich–Kwong equation is as follows:

```
>> state = 'L'; eos = 'rk';
>> [Z V] = cubicEOSZ(state,eos,T,P,Tc,Pc,w)
Z =
    0.0433
V =
    133.2663
```

Results of calculations for each equation of state are summarized in [Table 4.3](#).

TABLE 4.3
Results of Calculations (Molar Volume, cm³/mol)

State	Experimental Data ⁸	Equation of State			
		VDW	RK	SRK	PR
Saturated vapor	2482	2666.9	2555.3	2520.3	2486.4
Saturated liquid	115.0	190.981	133.266	127.813	112.602

4.2 THERMODYNAMIC PROPERTIES OF FLUIDS

4.2.1 ENTHALPY CHANGE

The heat capacity, C_p , depends on temperature and can be expressed as a function of temperature:

$$\frac{C_p}{R} = A + BT + CT^2 + DT^{-2}$$

For some materials C or D may be 0. The enthalpy change, ΔH , and the mean heat capacity, $\langle C_p \rangle_H$, is given by

$$\Delta H = Q = \int_{T_1}^{T_2} C_p dT, \quad C_{pH} = \frac{\Delta H}{T_2 - T_1}$$

The MATLAB function *delH* estimates ΔH and $\langle C_p \rangle_H$ using the coefficients of the heat capacity relation at the given temperature range.

```
function [Q, mc] = delH(C, T1, T2)
% Calculates enthalpy change and avg. heat capacity of pure material
% input:
%     C: coefficients of Cp relation (C=[A,B,C,D])
%     T1, T2: temperature range (lower and upper limits of integral)
% output:
%     Q: delta H
R = 8.314; % J/(mol-K)
fH = @(T) C(1) + C(2)*T + C(3)*T.^2 + C(4)*T.^(-2); %T(K)
intCp = quadl(fH, T1, T2); Q = R*intCp; % J
mc = intCp/(T2-T1); % (Cp)h/R
```

Example 4.4: Enthalpy Change of Methane⁹

Determine the mean heat capacity, $\langle C_p \rangle_H/R$, and the heat required to raise the temperature of 1 mol of methane from 260°C to 600°C in a steady flow process at a sufficiently low pressure that methane may be considered an ideal gas. For methane, the heat capacity is given by $C_p = 1.702 + 9.081 \times 10^{-3}T - 2.164 \times 10^{-6}T^2$.

Solution

The function *delH* produces desired outputs:

```
>> [Q mc] = delH([1.702 9.081e-3 -2.164e-6 0], 533.15, 873.15)
Q =
1.9778e+04
mc =
6.9965
```

4.2.2 DEPARTURE FUNCTION

The departure function implies the departure of the actual fluid property from the same ideal gas property at the same temperature and pressure. The changes in thermodynamic properties of an actual fluid are equal to those for an ideal gas undergoing the same change of state plus the departure from ideal gas behavior of the initial state. Once the fluid equation of state is known, the departure function can be evaluated.

The change in enthalpy is expressed as

$$\Delta H = H_2 - H_1 = (H_2 - H_2^{ig}) + (H_2^{ig} - H_1^{ig}) - (H_1 - H_1^{ig})$$

Here, $H_2 - H_2^{ig}$ and $H_1 - H_1^{ig}$ are calculated by using the departure function, and $H_2^{ig} - H_1^{ig}$ is given by

$$H_2^{ig} - H_1^{ig} = \int_{T_1}^{T_2} C_p dT = \int_{T_1}^{T_2} (A + BT + CT^2 + DT^3) dT$$

Similarly, the change in entropy is written as

$$\Delta S = S_2 - S_1 = (S_2 - S_2^{ig}) + (S_2^{ig} - S_1^{ig}) - (S_1 - S_1^{ig})$$

where $S_2^{ig} - S_1^{ig}$ is obtained from the following relationship:

$$S_2^{ig} - S_1^{ig} = \int_{T_1}^{T_2} \frac{C_p}{T} dT + R \ln \frac{P_2}{P_1} = \int_{T_1}^{T_2} \left(\frac{A}{T} + B + CT + DT^2 \right) dT + R \ln \frac{P_2}{P_1}$$

4.2.2.1 Departure Function from the Virial Equation of State

The virial equation of state may be used to represent the enthalpy and entropy departure functions:

$$\frac{H - H^{ig}}{RT} = -P_r \left[\frac{1.0972}{T_r^{2.6}} - \frac{0.083}{T_r} + w \left(\frac{0.8944}{T_r^{5.2}} - \frac{0.139}{T_r} \right) \right]$$

$$\frac{S - S^{ig}}{R} = -P_r \left(\frac{0.675}{T_r^{2.6}} + w \frac{0.722}{T_r^{5.2}} \right)$$

4.2.2.2 Departure Function from the VDW (van der Waals) Equation of State

Using the VDW (van der Waals) equation of state, we obtain the expressions for the enthalpy and entropy departure functions as

$$\frac{H - H^{ig}}{RT} = Z - 1 - \frac{3.375\beta}{ZT_r}, \quad \frac{S - S^{ig}}{R} = \ln(Z - \beta)$$

where $\beta = bP/RT$.

4.2.2.3 Departure Function from the RK (Redlich–Kwong) Equation of State

Using the RK (Redlich–Kwong) equation of state, we obtain the expressions for the enthalpy and entropy departure functions as

$$\frac{H - H^{ig}}{RT} = (Z - 1) - \frac{1.5\Psi}{\sigma T_r^{1.5}} \ln \left(1 + \frac{b}{V} \right)$$

$$\frac{S - S^{ig}}{R} = \ln \left(Z - \frac{bP}{RT} \right) - \frac{\Psi}{2\sigma T_r^{1.5}} \ln \left(1 + \frac{b}{V} \right)$$

4.2.2.4 Departure Function from the SRK (Soave–Redlich–Kwong) Equation of State

If the SRK (Soave–Redlich–Kwong) equation of state is used, the enthalpy and entropy departure functions are represented as

$$\frac{H - H^{ig}}{RT} = Z - 1 - \frac{A}{\beta} \left(\frac{\kappa \sqrt{T_r}}{1 + \kappa (1 - \sqrt{T_r})} + 1 \right) \ln \left(1 + \frac{\beta}{Z} \right)$$

$$\frac{S - S^{ig}}{R} = \ln(Z - \beta) - \frac{A}{\beta} \left(\frac{\kappa \sqrt{T_r}}{1 + \kappa (1 - \sqrt{T_r})} \right) \ln \left(1 + \frac{\beta}{Z} \right)$$

where $A = aP/R^2T^2$.

4.2.2.5 Departure Function from the PR (Peng–Robinson) Equation of State

The PR (Peng–Robinson) equation of state may be used to represent departure functions. The enthalpy departure function is given by

$$\frac{H - H^{ig}}{RT} = Z - 1 - \frac{A}{\beta \sqrt{8}} \left(1 + \frac{\kappa \sqrt{T_r}}{\sqrt{\alpha}} \right) \ln \left[\frac{Z + (1 + \sqrt{2})\beta}{Z + (1 - \sqrt{2})\beta} \right]$$

The entropy departure function is given by

$$\frac{S - S^{ig}}{R} = \ln(Z - \beta) - \frac{A}{\beta \sqrt{8}} \left(\frac{\kappa \sqrt{T_r}}{\sqrt{\alpha}} \right) \ln \left[\frac{Z + (1 + \sqrt{2})\beta}{Z + (1 - \sqrt{2})\beta} \right]$$

The internal energy departure function is given by

$$\frac{U - U^{ig}}{RT} = - \frac{A}{\beta \sqrt{8}} \frac{\kappa \sqrt{T_r}}{\sqrt{\alpha}} \ln \left[\frac{Z + (1 + \sqrt{2})\beta}{Z + (1 - \sqrt{2})\beta} \right]$$

The Gibbs free energy departure function is given by

$$\frac{G - G^{ig}}{RT} = Z - 1 - \ln(Z - \beta) - \frac{A}{\beta \sqrt{8}} \ln \left[\frac{Z + (1 + \sqrt{2})\beta}{Z + (1 - \sqrt{2})\beta} \right]$$

The MATLAB function *deptfun* calculates departure functions. The basic syntax is

```
[Z V dH dS] = deptfun(state,eos,T,P,Tc,Pc,w)
```

where state denotes the state of the fluid (liquid: L, vapor: V); eos is the equation of state being used (VR, VDW, RK, SRK, PR); T and P are the temperature (K) and pressure (bar), respectively; Tc and Pc are the critical temperature (K) and pressure (bar), respectively; w is the acentric factor; Z is the compressibility factor; V is the molar volume (cm³/mol); dH is the change in enthalpy (J/mol); dS is the change in entropy (J/mol/K).

```

function [Z V dH dS] = deptfun(state,eos,T,P,Tc,Pc,w)
% Calculation of departure functions using the virial and cubic EOS
% inputs
%   state: fluid state (liquid: L, vapor: V)
%   eos: type of the equation of state (VR, VDW, RK, SRK, PR)
%   T,P: temperature (K) and pressure (bar)
%   Tc,Pc: critical temperature (K) and critical pressure (bar)
%   w: acentric factor
% outputs:
%   Z: compressibility factor
%   V: molar volume
%   dH: enthalpy departure
%   dS: entropy departure

% Tr and Pr (reduced T and P)
Tr = T/Tc; Pr = P/Pc; R = 83.14; % cm^3*bar/mol/K

% Set parameters
eos = upper(eos);
switch eos
    case 'VDW'
        al = 1; sm = 0; ep = 0; om = 0.125; ps = 0.42188; kappa = 0;
    case 'RK'
        al = 1./sqrt(Tr); sm = 1; ep = 0; om = 0.08664; ps = 0.42748;
    case 'SRK'
        kappa = 0.480 + 1.574*w - 0.176*w^2;
        al = (1 + kappa*(1-sqrt(Tr))).^2;
        sm = 1; ep = 0; om = 0.08664; ps = 0.42748;
    otherwise % PR, or VR
        kappa = 0.37464 + 1.54226*w - 0.26992*w^2;
        al = (1 + kappa*(1-sqrt(Tr))).^2;
        sm = 1+sqrt(2); ep = 1-sqrt(2); om = 0.0778; ps = 0.45724;
    end

% compressibility factor (Z)
state = upper(state);
beta = om*Pr./Tr; q = ps*al./(om*Tr); % beta and q
if strcmp(eos,'VR') % virial (VR) EOS: vapor phase
    B0 = 0.083 - 0.422./(Tr.^1.6); B1 = 0.139 - 0.172./(Tr.^4.2);
    B = R*Tc.* (B0 + w.*B1)./Pc;
    Z = 1 + B.*P./(R*T);
else
    fV = @(Z) 1+beta-q*beta.* (Z-beta)./((Z+ep*beta).* (Z+sm*beta)) - Z;
    fL = @(Z) beta+(Z+ep*beta).* (Z+sm*beta).* (1+beta-Z)./(q.*beta) - Z;
    switch state
        case 'V'
            Z = fzero(fV, 1);
        case 'L'
            Z = fzero(fL, beta);
    end
end
V = Z*R*T/P; % cm^3/mol

% departure function
a = ps*al*R^2*Tc.^2/Pc; b = om*R*Tc./Pc; % a, b
Ad = a*P/(R^2*T^2); Bd = b*P/(R*T); % A, B

```

```

if strcmp(eos,'VR') % virial EOS: vapor phase
    dH = -Pr*(1.0972/Tr^2.6-0.083/Tr+w*(0.8944/Tr^5.2-0.139/Tr))*R*T;
    dS = -Pr*(0.675/Tr^2.6 + w*0.722/Tr^5.2)*R;
elseif strcmp(eos,'RK') % Redlich-Kwong EOS
    dH = (Z - 1 - 1.5*ps/(om*Tr^1.5) * log(1 + b/V))*R*T;
    dS = (log(Z-Bd) - 0.5*ps/(om*Tr^1.5) * log(1 + b/V))*R;
elseif strcmp(eos,'VDW') % van der Waals EOS
    dH = (Z - 1 - 3.375*Bd/Tr/Z)*R*T;
    dS = R*log(Z-Bd);
elseif strcmp(eos,'SRK') % Soave-Redlich-Kwong EOS
    dH = (Z - 1 + (-kappa*sqrt(Tr)/(1 + kappa*(1-sqrt(Tr))) - 1)*...
        (Ad/Bd)*log(1 + Bd/Z))*R*T;
    dS = (log(Z - Bd) - kappa*sqrt(Tr)/(1 + kappa*(1-sqrt(Tr)))*...
        (Ad/Bd)*log(1 + Bd/Z))*R;
elseif strcmp(eos,'PR') % Peng-Robinson EOS
    dH = (Z-1 - (Ad/(Bd*sqrt(8)))*(1+ kappa*sqrt(Tr)/sqrt(al))*...
        log((Z + sm*Bd)/(Z + ep*Bd)))*R*T;
    dS = (log(Z-Bd) - (Ad/(Bd*sqrt(8)))*(kappa*sqrt(Tr)/sqrt(al))*...
        log((Z + sm*Bd)/(Z + ep*Bd)))*R;
end
% R = 83.14 cm^3*bar/mol/K = 8.314 J/mol/K
dH = dH/10; dS = dS/10; % dH:J/mol, dS:J/mol/K

```

The MATLAB function *delHS* calculates changes in the enthalpy and entropy of pure fluids due to phase changes. This function uses the function *deptfun* to evaluate the departure function. The simple syntax is

```
[dH dS] = delHS(state,eos,T1,P1,T2,P2,cmp)
```

where state denotes the state of the fluid (liquid: L, vapor: V); eos is the equation of state being used (VR, VDW, RK, SRK, PR); T1, T2 and P1, P2 are temperature (K) and pressure (bar) at state 1 and 2, respectively; cmp is a structure containing the properties of pure fluids.

It has seven fields, which consist of cmp.A, cmp.B, cmp.C, cmp.D, cmp.Tc, cmp.Pc, and cmp.w. A, B, C, and D are the coefficients of Cp equations; Tc and Pc are the critical temperature (K) and pressure (bar), respectively; and w is the acentric factor. dH is the change in enthalpy (J/mol) and dS is the change in entropy (J/mol/K).

```

function [dH dS] = delHS(state,eos,T1,P1,T2,P2,cmp)
% delHS.m: calculates changes in H and S of pure fluids due to phase
change
% inputs
%     state: fluid state (liquid: L, vapor: V)
%     eos: type of the equation of state (VR, VDW, RK, SRK, PR)
%     T1,P1: temperature (K) and pressure (bar) at state 1
%     T2,P2: temperature (K) and pressure (bar) at state 2
%     cmp: structure containing the properties of pure fluids
%         (A, B, C, D (coefficients of Cp equation), Tc, Pc, w)
% outputs:
%     dH: change in enthalpy
%     dS: change in entropy

% Data of pure fluids
A = cmp.A; B = cmp.B; C = cmp.C; D = cmp.D;
Tc = cmp.Tc; Pc = cmp.Pc; w = cmp.w;

```

```

% changes in enthalpy(H) and entropy(S)
[Z1 V1 dH1 dS1] = deptfun(state,eos,T1,P1,Tc,Pc,w);
[Z2 V2 dH2 dS2] = deptfun(state,eos,T2,P2,Tc,Pc,w);

% phase change in idela gas
R = 8.314;
fH = @(T) A + B*T + C*T.^2 + D*T.^3;
fS = @(T) A./T + B + C*T + D*T.^2;
dHi = quadl(fH,T1,T2); dSi = quadl(fS,T1,T2);
dsi = quadl(fS,T1,T2) - R*log(P2./P1);

% changes in H and S due to phase change
dH = dH2 + dHi - dH1;
dS = dS2 + dSi - dS1;
end

```

Example 4.5: Enthalpy and Entropy Departure of *n*-Butane Gas^{10,11}

Determine the enthalpy departure $H^R = H - H^{ig}$ and the entropy departure $S^R = S - S^{ig}$ for *n*-butane gas at 50 bar and 500 K. For *n*-butane gas, $T_c = 425.1$ K, $P_c = 37.96$ bar, and $w = 0.2$.

Solution

Set state = 'v'. The script *resbutane* calls the function *deptfun* to calculate residual properties.

```

% resbutane.m: residual properties of n-butane
eosset = {'VR', 'VDW', 'RK', 'SRK', 'PR'};
Tc = 425.1; Pc = 37.96; w = 0.2; T = 500; P = 50; state = 'v';
for i = 1:length(eosset)
    eos = eosset{i};
    [Z V dH dS] = deptfun(state,eos,T,P,Tc,Pc,w);
    fprintf('The equation of state=%s: Z=%g H^R=%g S^R=%g\n', eos,Z,dH,dS);
end

```

The solution can now be obtained by executing the script *resbutane*.

```

>> resbutane
The equation of state = VR : Z = 0.740084 H^R = -3845.11 S^R = -5.52747
The equation of state = VDW : Z = 0.660991 H^R = -3935.39 S^R = -5.42065
The equation of state = RK : Z = 0.685188 H^R = -4502.78 S^R = -6.54207
The equation of state = SRK : Z = 0.722389 H^R = -4821.29 S^R = -7.40783
The equation of state = PR : Z = 0.690914 H^R = -4984.72 S^R = -7.42094

```

Example 4.6: Enthalpy and Entropy Departures of Propane Gas¹²

Propane gas undergoes a change of state from an initial condition of 5 bar and 105°C (state 1) to 25 bar and 190°C (state 2).

1. Determine the enthalpy departure $H - H^{ig}$ and the entropy departure $S - S^{ig}$ at each state from the Peng–Robinson equation.
2. Calculate changes in enthalpy and entropy for a change from state 1 to state 2. For propane gas, $T_c = 369.8$ K, $P_c = 4.249$ MPa (=42.49 bar), and $w = 0.152$. The heat capacity coefficients are given by $A = -4.224$, $B = 0.3063$, $C = -1.586 \times 10^{-4}$, and $D = 3.215 \times 10^{-8}$.

TABLE 4.4
Enthalpy and Entropy Departures at Each State

	State 1		State 2	
	$H - H^{ig}$ (J/mol)	$S - S^{ig}$ (J/mol/K)	$H - H^{ig}$ (J/mol)	$S - S^{ig}$ (J/mol/K)
Reference 13	-400.512	-0.7083	-1489.9	-2.2925
Results	-400.496	-0.7082	-1489.8	-2.2923

Solution

1. Set state = 'v' and eos = 'pr' (Peng–Robinson equation). The following commands produce results at each state:

```
>> Tc = 369.8; PC = 42.49; w = 0.152; T = 378.15; P = 5; eos = 'pr';
state = 'v';
>> T1 = 378.15; P1 = 5; T2 = 463.15; P2 = 25;
>> [Z1 V1 dH1 dS1] = deptfun(state,eos,T1,P1,Tc,PC,w)
Z1 =
0.9574
V1 =
6.0199e+03
dH1 =
-400.4959
dS1 =
-0.7082
>> [Z2 V2 dH2 dS2] = deptfun(state,eos,T2,P2,Tc,PC,w)
Z2 =
0.8891
V2 =
1.3694e+03
dH2 =
-1.4898e+03
dS2 =
-2.2923
```

Results are summarized in Table 4.4.

2. The following commands produce desired outputs:

```
>> cmp.A=-4.224; cmp.B=0.3063; cmp.C=-1.586e-4; cmp.D=3.215e-8;
>> cmp.Tc=369.8; cmp.Pc=42.49; cmp.w=0.152;
>> eos = 'pr'; state = 'v'; T1 = 378.15; P1 = 5; T2 = 463.15; P2 = 25;
>> [dH dS] = delHS(state,eos,T1,P1,T2,P2,cmp)
dH =
7.3155e+03
dS =
5.0285
```

We can see that $\Delta H = 7315.5$ J/mol and $\Delta S = 5.0285$ J/(mol·K).

4.2.3 ENTHALPY OF MIXTURE

The mixing rules and the combining rule for the calculation of a and b are

$$a = \sum_i \sum_j x_i x_j a_{ij}, \quad a_{ij} = \sqrt{a_i a_j} (1 - k_{ij}), \quad b = \sum_i x_i b_i$$

where

k_{ij} is a binary interaction parameter specific to an $i-j$ molecular pair
 a_{ij} and b_i are the parameters for pure component i

The enthalpy of a mixture is given by

$$H = H_{ig}^0 + \Delta H$$

where H_{ig}^0 can be obtained from the relation

$$H_{ig}^0 = \int_{T_0}^T C_{pV}^0 dT = \sum_{i=1}^5 \frac{a_k}{k} (T^k - T_0^k)$$

Here, values of a_i ($i = 1, \dots, 5$) can be found elsewhere.¹⁴

The MATLAB function *khmix* estimates the enthalpy of a mixture. In this function, the RK (Redlich–Kwong), SRK (Soave–Redlich–Kwong), and PR (Peng–Robinson) equations can be used as the equation of state. This function has the syntax

```
[Z H] = khmix(x, P, T, state, eos, mfp)
```

where x is the mole fractions of all components (column vector), P is the pressure (Pa), T is the temperature (K), $state$ denotes the state of the fluid ('L': liquid, 'V': vapor), eos is the equation of state being used ('RK', 'SRK', or 'PR'), mfp is a structure containing properties of all components, Z is the compressibility factor, H is the enthalpy of the mixture (J/mol).

Structure mfp has several fields: $mfp.Pc$ and $mfp.Tc$ for critical pressure (Pa, column vector) and critical temperature (K, column vector), $mfp.w$ for acentric factors of all components (column vector), $mfp.k$ for binary interaction parameter matrix ($n \times n$ symmetric matrix), and $mfp.Afi$ for coefficients of the ideal gas heat capacity relation (number of components x number of constants matrix) (J/mol/K).

```
function [Z H] = khmix(x, P, T, state, eos, mfp)
% Estimation of enthalpy of mixture
% Inputs:
%   x: mole fractions of all components (column vector)
%   P, T: pressure(Pa) and temperature(K)
%   state: fluid state('L': liquid, 'V': vapor)
%   eos: equation of state('RK', 'SRK', or 'PR')
%   mfp: structure with property fields
%     mfp.Pc, mfp.Tc: critical P(Pa) and T(K) (column vector)
%     mfp.w: acentric factors of all components (column vector)
%     mfp.k: binary interaction parameter matrix(n x n symmetric)
%     mfp.Afi: coefficients of ideal gas heat capacity relation(J/mol/K)
% Outputs:
%   Z: compressibility factor
%   H: enthalpy of mixture (J/mol)
Pc = mfp.Pc; Tc = mfp.Tc; w = mfp.w; k = mfp.k; % T, P: K, Pa
Afi = mfp.Afi; % btu/lbmole: 1btu/lbmole = 2.326 J/mol
R = 8.314; % gas constant: m^3 Pa/(mol K) = J/mol-K
Tr = T./Tc; Pr = P./Pc; nc = length(x);
eos = upper(eos); state = upper(state);
switch eos
    case{'RK'}
        ai = sqrt(0.4278./ (Pc.*Tr.^2.5)); bi = 0.0867./ (Pc.*Tr);
        A = sum(x.*ai); B = sum(x.*bi);
        Z = roots([1 -1 B*P*(A^2/B-B*P-1) -A^2*(B*P)^2/B]);
```

```

case{'SRK'}
    mx = 0.48+1.574*w-0.176*w.^2;
    al = (1+mx.* (1-sqrt(Tr))).^2;
    ai = 0.42747*al.*Pr./ (Tr.^2); bi = 0.08664*Pr./Tr;
    am = sqrt(ai'*ai).* (1-k);
    A = x*am*x'; B = sum(x.*bi);
    Z = roots([1 -1 A-B-B.^2 -A*B]);
case{'PR'}
    mx = 0.37464+1.54226*w-0.26992*w.^2;
    al = (1+mx.* (1-sqrt(Tr))).^2;
    ai = 0.45723553*al.*Pr./ (Tr.^2); bi = 0.0777961*Pr./Tr;
    am = sqrt(ai'*ai).* (1-k);
    A = x*am*x'; B = sum(x.*bi);
    Z = roots([1 B-1 A-3*B.^2-2*B B.^3+B.^2-A*B]);
end
iz = abs(imag(Z)); Z(and(iz>0,iz<=1e-6)) = real(Z(and(iz>0,iz<=1e-6)));
for i = 1:length(Z), zind(i) = isreal(Z(i)); end
Z = Z(zind);
if state == 'L'
    Z = min(Z);
else
    Z = max(Z);
end
V = R*T*Z/P; % m^3/mol

% Hv0 = int(Cp)
Tf = (T-273.15)*1.8 + 32; % T: K->F
Hv0 = x*(Afi(:,1)*Tf + Afi(:,2)*Tf.^2/2 + Afi(:,3)*Tf.^3/3 + ...
    Afi(:,4)*Tf.^4/4 + Afi(:,5)*Tf.^5/5);
Hv0 = Hv0*2.326; % Btu/lbmole -> J/mol

% compute enthalpy
switch eos
    case{'RK'}
        H = Hv0 + R*T*(Z - 1 - 3*(A.^2)*log(1 + B*P/Z)/(2*B));
    case{'SRK'}
        hsum = 0;
        for i = 1:nc
            for j = 1:nc
                hsum = hsum + x(i)*x(j)*am(i,j)*(1 - ...
                    mx(i)*sqrt(Tr(i))/(2*sqrt(al(i))) - ...
                    mx(j)*sqrt(Tr(j))/(2*sqrt(al(j))));
            end
        end
        H = Hv0 + R*T*(Z - 1 - log((Z + B)/Z)*hsum/B);
    case{'PR'}
        hsum = 0;
        for i = 1:nc
            for j = 1:nc
                hsum = hsum + x(i)*x(j)*am(i,j)*(1 - ...
                    mx(i)*sqrt(Tr(i))/(2*sqrt(al(i))) - ...
                    mx(j)*sqrt(Tr(j))/(2*sqrt(al(j))));
            end
        end
        H = Hv0 + R*T*(Z - 1 - log((Z + B)/Z)*hsum/B);
    end
end

```

Example 4.7: Enthalpy of Mixture

Estimate the liquid-phase enthalpy of the mixture of methane(1)/ethane(2)/propane(3) at -158 K and 6.8947 bar . The liquid-phase mole fractions of components are $x_1 = 0.419$, $x_2 = 0.3783$, $x_3 = 0.2027$, and the properties of each component are shown in [Table 4.5](#) (ΔH_f , ΔG_f : J/mol).

[Table 4.6](#) shows coefficients of the ideal gas heat capacity relation for each component.

Solution

The script `usekhemix` specifies required data and calls the function `khemix` to perform calculations.

```
% usekhemix.m
state = 'L'; nx = [0.419 0.3783 0.2027];
P = 6.8947e5; T = 158; eos = 'rk';
mfp.Pc = [45.99 48.72 42.48]*1e5; mfp.Tc = [190.6 305.3 369.8];
mfp.w = [0.012 0.1 0.152]; mfp.k = zeros(3,3);
mfp.Afi = [8.245223 0.3806333e-2 0.8864745e-5 -0.7461153e-8 0.182296e-11;
11.51606 0.140309e-1 0.854034e-5 -0.1106078e-7 0.31622e-11;
15.58683 0.2504953e-1 0.1404258e-4 -0.352626e-7 0.1864467e-10];
state = upper(state); if state == 'L', x = nx; else x = ny; end
[Z H] = khemix(x,P,T,state,eos,mfp);
fprintf('Equation of state: %s, State: %s',upper(eos),upper(state));
fprintf('\nCompressibility factor = %g, Enthalpy = %g (J/mol)\n',Z,H);
```

Results of execution of the script `usekhemix` are

```
>> usekhemix
Equation of state: RK, State: L
Compressibility factor = 0.027695, Enthalpy = -18474.5 (J/mol)
```

[Table 4.7](#) summarizes results for each equation of state.

TABLE 4.5
Properties of Each Component of the Mixture

Component	x	w	$T_c\text{ (K)}$	$P_c\text{ (bar)}$	ΔH_f	ΔG_f
Methane	0.4190	0.012	190.6	45.99	-74,520	-50,460
Ethane	0.3783	0.100	305.3	48.72	-83,820	-31,855
Propane	0.2027	0.152	369.8	42.48	-104,680	-24,290

Source: Smith, J.M. et al., *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, 2005, pp. 680–686.

TABLE 4.6
Coefficients of the Ideal Gas Heat Capacity Relation

Component	a_1	$a_2 \times 10^2$	$a_3 \times 10^5$	$a_4 \times 10^8$	$a_5 \times 10^{11}$
Methane	8.245223	0.380633	0.8864745	-0.746115	0.182296
Ethane	11.51606	1.40309	0.854034	-1.106078	0.31622
Propane	15.58683	2.504953	1.404258	-3.52626	1.864467

TABLE 4.7
Estimated Enthalpy and Z by Different Equations of State

Equation of State	Compressibility Factor	Enthalpy
<i>RK</i>	0.0276950	-18,474.5
<i>SRK</i>	0.0276512	-10,664.5
<i>PR</i>	0.0245126	-12,089.2

4.3 FUGACITY COEFFICIENT

4.3.1 FUGACITY COEFFICIENTS OF PURE SPECIES

The fugacity of a pure component, f , is defined as

$$f = \phi P$$

The fugacity coefficient ϕ is determined first followed by multiplication with pressure to give f .

4.3.1.1 Fugacity Coefficient from the Virial Equation of State

The second virial coefficient is a function of temperature only for a pure species. At low to moderate pressure, the virial equation of state truncated after the second virial coefficient can be used:

$$\ln \phi = \frac{BP}{RT} = \frac{P_r}{T_r} \left(B^0 + wB^1 \right)$$

4.3.1.2 Fugacity Coefficient from the van der Waals Equation of State

By using the van der Waals equation of state, we have

$$\ln \phi = Z - 1 - \ln \left\{ Z \left(1 - \frac{b}{V} \right) \right\} - \frac{a}{RTV}$$

4.3.1.3 Fugacity Coefficient from the Redlich–Kwong Equation of State

The Redlich–Kwong equation of state provides the following description for the fugacity coefficient:

$$\ln \phi = Z - 1 - \ln \left\{ Z \left(1 - \frac{b}{V} \right) \right\} - \frac{a}{bRT} \ln \left(1 + \frac{b}{V} \right)$$

4.3.1.4 Fugacity Coefficient from the Soave–Redlich–Kwong and Peng–Robinson Equations of State

For hydrocarbons and simple gases, the Soave–Redlich–Kwong equation and the Peng–Robinson equation provide a more accurate description:

$$\ln \phi = Z - 1 - \ln (Z - B) - \frac{a(T)}{bRT(\sigma - \epsilon)} \ln \left[\frac{Z + \sigma B}{Z + \epsilon B} \right]$$

where $B = bP/RT$ and σ and ϵ are the parameters of the equation of state.⁶

4.3.1.5 Fugacity Coefficient from Experimental Data

When sufficient experimental data on the compressibility factor (Z) and pressure (P) are available, the fugacity coefficient can be obtained from the numerical integration:

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

The MATLAB function *phigas* estimates the fugacity coefficient using the virial and cubic equations of state. The basic syntax is

```
[phig f] = phigas(state,eos,T,P,Tc,Pc,w)
```

where state denotes the state of the fluid (liquid: L, vapor: V); eos is the equation of state being used (VR, VDW, RK, SRK, PR); T and P are the temperature (K) and pressure (bar), respectively; Tc and Pc are the critical temperature (K) and pressure (bar), respectively; w is the acentric factor; phig is the fugacity factor; f is the fugacity (bar).

```
function [phig f] = phigas(state,eos,T,P,Tc,Pc,w)
% Estimates the fugacity coefficient using the virial and cubic EOS.
% input
% state: fluid state('L': liquid, 'V': vapor)
% eos: equation of state (VR, VDW, RK, SRK, PR)
% P,T: pressure(bar) and temperature(K)
% Tc,Pc: critical T(K) and P(bar)
% w: acentric factor
% output:
% phig: fugacity coefficient
% f: fugacity (bar)

% Tr and Pr (reduced T and P)
Tr = T/Tc; Pr = P/Pc; R = 83.14; % cm^3*bar/mol/K

% Set parameters.
eos = upper(eos);
switch eos
    case 'VDW'
        al = 1; sm = 0; ep = 0; om = 0.125; ps = 0.42188; kappa = 0;
    case 'RK'
        al = 1./sqrt(Tr); sm = 1; ep = 0; om = 0.08664; ps = 0.42748;
    case 'SRK'
        kappa = 0.480 + 1.574*w - 0.176*w^2;
        al = (1 + kappa*(1-sqrt(Tr))).^2;
        sm = 1; ep = 0; om = 0.08664; ps = 0.42748;
    otherwise % PR or VR
        kappa = 0.37464 + 1.54226*w - 0.26992*w^2;
        al = (1 + kappa*(1-sqrt(Tr))).^2;
        sm = 1+sqrt(2); ep = 1-sqrt(2); om = 0.0778; ps = 0.45724;
end

% compressibility factor (Z)
state = upper(state);
beta = om*Pr./Tr; q = ps*al./(om*Tr); % beta and q
if strcmp(eos,'VR') % virial EOS: vapor phase
    B0 = 0.083 - 0.422./(Tr.^1.6); B1 = 0.139 - 0.172./(Tr.^4.2);
    B = R*Tc.* (B0 + w.*B1)./Pc;
    Z = 1 + B.*P./ (R*T);
```

```

else
    fV = @(Z) 1+beta-q*beta.*(Z-beta)./((Z+ep*beta).*(Z+sm*beta)) - Z;
    fL = @(Z) beta+(Z+ep*beta).*(Z+sm*beta).* (1+beta-Z)./(q.*beta) - Z;
    switch state
        case 'V'
            Z = fzero(fV, 1);
        case 'L'
            Z = fzero(fL, beta);
    end
end
V = Z*R*T/P; % cm^3/mol

% fugacity coefficient
a = ps*al*R^2*Tc.^2/Pc; b = om*R*Tc./Pc; % a, b
qi = a/(b*R*T); Bd = b*P/(R*T); % A, B
if strcmp(eos,'VR') % virial EOS: vapor phase
    phig = exp(Pr*(B0 + w*B1)./Tr);
elseif strcmp(eos,'VDW')
    phig = exp(Z - 1 - log(Z.*(1 - b/V)) - a./(R*T*V));
elseif strcmp(eos,'RK')
    phig = exp(Z-1 - log(Z.*(1 - b/V)) - (a/(b*R*T)) * log(1+b/V));
else % SRK or PR
    phig = exp(Z-1 - log(Z-Bd) - (qi/(sm-ep))*...
        log((Z + sm*Bd)/(Z + ep*Bd)));
end
f = phig*P; % fugacity (bar)
end

```

Example 4.8: Fugacity for Acetylene Gas¹⁶

Find the fugacity for acetylene at 250 K and 10 bar. For acetylene, $T_c = 308.3$ K, $P_c = 6.139$ MPa (=61.39 bar), and $w = 0.187$.

Solution

Using the Soave–Redlich–Kwong (SRK) equation, we obtain the following results:

```

>> Tc = 308.3; Pc = 61.39; w = 0.187; T = 250; P = 10; eos = 'srk'; state = 'v';
>> [phig f] = phigas(state,eos,T,P,Tc,Pc,w)
phig =
    0.8956
f =
    8.9559

```

Table 4.8 shows results obtained using the virial and cubic equations of state.

TABLE 4.8
Estimated Fugacities and Fugacity Coefficients

Equation of State	VR	VDW	RK	SRK	PR
Fugacity coefficient	0.8938	0.9208	0.9007	0.8956	0.8889
Fugacity (bar)	8.9383	9.2075	9.0071	8.9559	8.8893

4.3.2 FUGACITY COEFFICIENT OF A SPECIES IN A MIXTURE

The fugacity coefficient of species i in a mixture, $\hat{\phi}_i$, is defined as

$$\hat{\phi}_i = \frac{\hat{f}_i}{y_i P}$$

4.3.2.1 Fugacity Coefficient from the Virial Equation of State

From the virial equation of state, the fugacity coefficient of component i in a mixture, $\hat{\phi}_i = \hat{f}_i/y_i P$, is given by¹⁷

$$\ln \hat{\phi}_i = \frac{P}{RT} \left(2 \sum_j y_j B_{ji} - B \right), \quad B = \sum_k \sum_j y_k y_j B_{kj}$$

For a binary mixture, we have

$$\ln \hat{\phi}_1 = \frac{P}{RT} (2y_1 B_{11} + 2y_2 B_{12} - B), \quad \ln \hat{\phi}_2 = \frac{P}{RT} (2y_1 B_{12} + 2y_2 B_{22} - B)$$

4.3.2.2 Fugacity Coefficient from the Cubic Equations of State

For the i component in a mixture, let

$$a_i(T) = \Psi \frac{\alpha(T_{ri}) R^2 T_{ci}^2}{P_{ci}}, \quad b_i = \Omega \frac{RT_{ci}}{P_{ci}}, \quad \beta_i = \frac{b_i P}{RT}, \quad q_i = \frac{a_i(T)}{b_i RT}$$

and use the mixing rule to obtain the parameters a and b . The fugacity coefficient of component i in a mixture is expressed as¹⁸

$$\ln \hat{\phi}_i = \frac{b_i}{b} (Z - 1) - \ln(Z - \beta) - \bar{q}_i I$$

where

$$I = \frac{1}{\sigma - \epsilon} \ln \left(\frac{Z + \sigma \beta}{Z + \epsilon \beta} \right), \quad q = \frac{a}{bRT}, \quad \bar{q}_i = q \left(1 + \frac{\bar{a}_i}{a} - \frac{b_i}{b} \right), \quad \bar{a}_i = \frac{\partial(na)}{\partial n_i} \Big|_{T, n_j}$$

If we use the Peng–Robinson equation, the equation for $\ln \hat{\phi}_i$ is written as

$$\ln \hat{\phi}_i = -\ln(Z - \beta) + \frac{\beta_i}{\beta} (Z - 1) - \frac{q}{\sqrt{8}} \left(\frac{2}{a} \sum_j x_j a_{ij} - \frac{\beta_i}{\beta} \right) \ln \left(\frac{Z + (1 + \sqrt{2})\beta}{Z + (1 - \sqrt{2})\beta} \right)$$

4.3.2.3 Fugacity Coefficient from the van der Waals Equation of State

From the van der Waals equation of state, we have the following description for the fugacity coefficient of component i in a mixture:

$$\ln \hat{\phi}_i = -\ln(Z - \beta) + \frac{\beta_i}{Z - \beta} - \frac{2}{Z} \sum_j x_j A_{ij}, \quad A_{ij} = \frac{a_{ij} P}{R^2 T^2} = \frac{a_{ij} \beta}{bRT}$$

The MATLAB function *phimix* estimates the fugacity coefficients of all components in a mixture from cubic equations of state. This function has the syntax

```
[Z,V,phi] = phimix(ni,P,T,Pc,Tc,w,k,state,eos)
```

where *ni* is the number of moles (or mole fractions) of each component in a mixture (vector), *P* is the pressure (Pa), *T* is the temperature (K), *w* is the acentric factor, *k* is a matrix of binary interaction parameters, *state* denotes the state of the fluid ('L': liquid, 'V': vapor), *eos* is the equation of state being used ('RK', 'SRK', or 'PR'), *Z* is the compressibility factor, *V* is the molar volume, and *phi* is the fugacity coefficients of all components (vector).

```
function [Z,V,phi] = phimix(ni,P,T,Pc,Tc,w,k,state,eos)
% Estimates fugacity coefficients of all components in a mixture using
% the cubic equation of state
% Inputs:
%   ni: number of moles (or mole fractions) of each component (vector)
%   P, T: pressure(Pa) and temperature(K)
%   Pc, Tc: critical P(Pa) and T(K) of all components (vector)
%   w: acentric factors of all components (vector)
%   k: symmetric matrix of binary interaction parameters (n x n)
%   state: fluid state('L': liquid, 'V': vapor)
%   eos: equation of state('RK', 'SRK', or 'PR')
% Outputs:
%   V: molar volume (m3/mol)
%   Z: compressibility factor
%   phi: fugacity coefficient vector

ni = ni(:); Pc = Pc(:); Tc = Tc(:); w = w(:); % column vector
x = ni/sum(ni); % mole fraction
R = 8.314; % gas constant: m^3 Pa/(mol K) = J/mol-K
Tref = 298.15; % reference T(K)
Pref = 1e5; % reference P(Pa)
Tr = T./Tc;
eos = upper(eos); state = upper(state);
switch eos
    case{'RK'}
        ep = 0; sm = 1; om = 0.08664; ps = 0.42748;
        al = 1./sqrt(Tr);
    case{'SRK'}
        ep = 0; sm = 1; om = 0.08664; ps = 0.42748;
        al = (1+(0.48+1.574*w-0.176*w.^2).* (1-sqrt(Tr))).^2;
    case{'PR'}
        ep = 1 - sqrt(2); sm = 1 + sqrt(2); om = 0.07780; ps = 0.45724;
        al = (1+(0.37464+1.54226*w-0.26992*w.^2).* (1-sqrt(Tr))).^2;
end
ai = ps*(R.^2).*al.* (Tc.^2)./Pc; am = sqrt(ai*ai').*(1 - k); % nxn matrix
a = x'*am*x; bi = om*R*Tc./Pc; b = x'*bi;
beta = b*P/R/T; q = a/(b*R*T);

% compressibility factor(Z) and molar volume (V)
state = upper(state);
c(1) = 1;
c(2) = (sm +ep)*beta - (1+beta);
c(3) = beta*(q + ep*sm*beta -(1+beta)*(sm+ep));
c(4) = -beta.^2*(q +(1+beta)*ep*sm);
```

```

% Roots
Z = roots(c);
iz = abs(imag(Z)); Z(and(iz>0,iz<=1e-6)) = real(Z(and(iz>0,iz<=1e-6)));
for i = 1:length(Z), zind(i) = isreal(Z(i)); end
Z = Z(zind);
if state == 'L'
    Z = min(Z);
else
    Z = max(Z);
end
V = R*T*Z/P;

% fugacity coefficients
bara = (2*ni'*am - a*ones(1,length(ni)))'; barb = bi;
phi = exp((Z - 1)*barb/b - log((V - b)*Z/V) + (a/(b*R*T))/(ep - sm)*...
    log((V + sm*b)/(V + ep*b))*(1 + bara/a - barb/b));
end

```

Example 4.9: Fugacity Coefficients in a Mixture¹⁹

Determine the fugacity coefficients of all components in a nitrogen(1)/methane(2) mixture by the Peng–Robinson equation at 100 K and 0.4119 MPa (4.119 bar). In this mixture, the mole fraction of nitrogen is $y_1 = 0.958$. For nitrogen, $T_{c1} = 126.1$ K, $P_{c1} = 3.394$ MPa (33.94 bar), and $\omega_1 = 0.04$, and for methane, $T_{c2} = 190.6$ K, $P_{c2} = 4.604$ MPa (46.04 bar), and $\omega_2 = 0.011$.

Solution

The following commands produce a vector of fugacity coefficients:

```

>> state = 'v'; k = zeros(2,2); eos='pr'; T = 100; P = 4.119e5; ni = [0.958
    0.042];
>> Tc = [126.1 190.6]; Pc = [33.94 46.04]*1e5; w = [0.04 0.011];
>> [Z,V,phi] = phimix(ni,P,T,Pc,Tc,w,k,state,eos)
Z =
    0.9059
V =
    0.0018
phi =
    0.9162
    0.8473

```

4.4 VAPOR PRESSURE

4.4.1 ANTOINE EQUATION

The Antoine equation is well known as a simple form of the shortcut vapor pressure equation:

$$\ln P_v = A - \frac{B}{T+C}$$

where A , B , and C are parameters. Multiplying both sides by $(T + C)$ to convert into a linear form followed by rearrangement, we have

$$y = a_1 + a_2 x_1 + a_3 x_2$$

where $y = \ln P_v$, $x_1 = 1/T$, $x_2 = \ln P_v/T$, $a_1 = A$, $a_2 = AC - B$, $a_3 = -C$.

4.4.2 RIEDEL EQUATION

The Riedel equation for vapor pressure is expressed as^{20,21}

$$\ln P_v = A + \frac{B}{T} + C \ln T + DT^6$$

where A , B , C , and D are parameters. This equation can be written in a linear form as

$$y = A + Bx_1 + Cx_2 + Dx_3$$

where $y = \ln P_v$, $x_1 = 1/T$, $x_2 = \ln T$, $x_3 = T^6$.

4.4.3 HARLECHER–BRAUN EQUATION²²

The Harlecher–Braun equation is reasonably accurate from low vapor pressure up to the critical pressure:

$$\ln P_v = A + \frac{B}{T} + C \ln T + D \frac{P_v}{T^2}$$

where A , B , C , and D are parameters. This equation can be expressed as a linear form:

$$y = A + Bx_1 + Cx_2 + Dx_3$$

where $y = \ln P_v$, $x_1 = 1/T$, $x_2 = \ln T$, $x_3 = P_v/T^2$.

Example 4.10: Vapor Pressure of 2,2,4-Trimethylpentane²³

Table 4.9 shows the vapor pressure of liquid 2,2,4-trimethylpentane at various temperatures. Use the data given in Table 4.9 to determine the parameters for the Antoine equation, the Riedel equation, and the Harlecher–Braun equation.

Solution

For the Antoine equation, let

$$y = \begin{bmatrix} \ln P_{v1} \\ \ln P_{v2} \\ \vdots \\ \ln P_{vn} \end{bmatrix}, \quad X = \begin{bmatrix} 1 & 1/T_1 & \ln P_{v1}/T_1 \\ 1 & 1/T_2 & \ln P_{v2}/T_2 \\ \vdots & \vdots & \vdots \\ 1 & 1/T_n & \ln P_{vn}/T_n \end{bmatrix}, \quad b = \begin{bmatrix} A \\ AC - B \\ -C \end{bmatrix}$$

and use the relation

$$b = (X^T X)^{-1} X^T y$$

TABLE 4.9
Vapor Pressure (P_v) of Liquid 2,2,4-Trimethylpentane

T (°C)	-15	-4.3	7.5	20.7	29.1	40.7	58.1	78.0	99.2
P_v (kPa)	0.667	1.333	2.666	5.333	8.000	13.33	26.66	53.33	101.32

to calculate A , B , and C . For the Riedel and the Harlecher–Braun equations, define suitably the matrix X and find the parameter vector $b = [A \ B \ C \ D]^T$. The MATLAB script *compVPeqn* calculates the parameters of these equations. For the Harlecher–Braun equation, the nonlinear equation

$$A + \frac{B}{T} + C \ln T + \frac{Dx}{T^2} - \ln x = 0$$

is solved at a given T to give the vapor pressure x .

```
% compVPeqn: comparison of vapor equations
clear all; clc;
T = [-15 -4.3 7.5 20.7 29.1 40.7 58.1 78.0 99.2] + 273.15;
Pv = [0.667 1.333 2.666 5.333 8.000 13.33 26.66 53.33 101.32];
Xa = [ones(1,length(T)); 1./T; log(Pv)./T]';
ba = inv(Xa'*Xa)*Xa'*log(Pv)';
Aa = ba(1); Ca = -ba(3); Ba = Aa*Ca-ba(2); % Antoine eq.
Xr = [ones(1,length(T)); 1./T; log(T); T.^6]';
br = inv(Xr'*Xr)*Xr'*log(Pv)';
Ar = br(1); Br = br(2); Cr = br(3); Dr = br(4); % Riedel eq.
Xh = [ones(1,length(T)); 1./T; log(T); Pv./T.^2]';
bh = inv(Xh'*Xh)*Xh'*log(Pv)';
Ah = bh(1); Bh = bh(2); Ch = bh(3); Dh = bh(4); % Harlecher-Braun eq
% plot
Ti = T(1):T(end);
Pa = exp(Aa - Ba./(Ti+Ca));
Pr = exp(Ar + Br./Ti + Cr*log(Ti) + Dr*Ti.^6);
Ph = [];
for k = 1:length(Ti)
    P0 = 100/length(Ti) * k;
    Phf = @(x) Ah + Bh/Ti(k) + Ch*log(Ti(k)) + Dh*x/Ti(k)^2 - log(x);
    Phv = fzero(Phf,P0); Ph = [Ph Phv];
end
plot(Ti,Pa,Ti,Pr,':',Ti,Ph,'.-',T,Pv,'o')
xlabel('T(C)'), ylabel('Pv(mmHg)')
legend('Antoine','Riedel','Harlecher-Braun','Data')
```

The script *compVPeqn* generates a plot illustrated in Figure 4.1 showing vapor pressure values obtained from various equations whose parameters are estimated based on the data given in Table 4.9.

```
>> compVPeqn
```

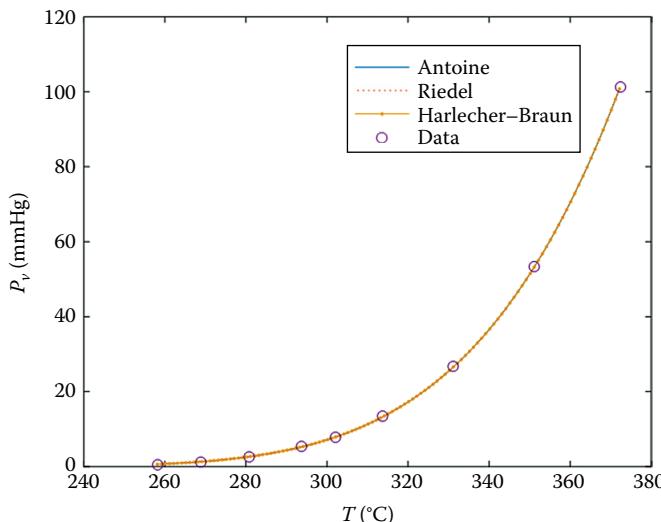


FIGURE 4.1 Vapor pressure of 2,2,4-trimethylpentane.

4.5 ACTIVITY COEFFICIENT

The Gibbs–Duhem equation may be used as the basis for estimating the vapor-phase composition and the activity coefficients. For a binary mixture, this equation can be expressed as

$$x_1 \left(\frac{\partial \ln \gamma_1}{\partial x_1} \right)_{P,T} + x_2 \left(\frac{\partial \ln \gamma_2}{\partial x_1} \right)_{P,T} = 0$$

where

- x_i is the mole fraction of component i in the liquid phase
- γ_i is the liquid-phase activity coefficient of component i

The activity coefficient of component i is defined as

$$\gamma_i = \frac{y_i P}{x_i P_i^v}$$

where

- y_i is the mole fraction of component i in the vapor phase
- P is the total pressure
- P_i^v is the vapor pressure of component i

The combination of the above two relations followed by some manipulations yields

$$\frac{dy_1}{dx_1} = \frac{y_1(1-y_1)}{y_1 - x_1} \frac{d \ln P}{dx_1}$$

If $P - x_1$ data are available, P can be expressed as a polynomial function of x_1 and the derivative $d \ln P / dx_1$ can be written as a function of x_1 . Integration of this relation from $x_1 = 0$ to $x_1 = 1$ gives y_1 . At the initial condition ($x_1 = 0, y_1 = 0$), the denominator of the derivative term becomes zero and l'Hopital's rule should be employed to give the following relation to be used in the integration:

$$\frac{dy_1}{dx_1} \approx \frac{\Delta y_1}{\Delta x_1} \approx \frac{\epsilon_{y_1}}{\epsilon_{x_1}} \approx (1 - 2y_1) \left(\frac{d \ln P}{dx_1} \right) \Big|_{x_1=0, y_1=0}$$

From this relation, $\epsilon_{x_1} = (d \ln P / dx_1) \Big|_{x_1=0} \cdot \epsilon_{x_1}$ is evaluated and used as the initial value for y_1 . As for ϵ_{x_1} , a very small value (e.g., 0.00001) may be used.

Example 4.11: Vapor-Phase Composition of Benzene/Acetic Acid System²⁴

Data of liquid-phase composition versus total pressure for the benzene(1)/acetic acid(2) system at 50°C are presented in [Table 4.10](#). Use the Gibbs–Duhem equation to estimate the composition of the vapor phase and the activity coefficients.

Solution

From the data, we can see that the vapor pressure of benzene is $P_1^v = 271$ mmHg and that of acetic acid is $P_2^v = 57.52$ mmHg. The MATLAB built-in function *polyfit* may be used in the regression of

TABLE 4.10
Total Pressure for the Benzene(1)/Acetic Acid(2)
System at 50°C

x_1	P (mmHg)	x_1	P (mmHg)
0.0	57.52	0.8286	250.20
0.0069	58.2	0.8862	259.00
0.1565	126.00	0.9165	261.11
0.3396	175.30	0.9561	264.45
0.4666	189.50	0.9840	266.53
0.6004	224.30	1.0	271.00
0.7021	236.00		

Source: Washburn, E.W. (ed.) et al., *Internal Critical Tables*, 1st ed., Vol. III, McGraw-Hill, New York, NY, 1928, p. 287.

the total pressure data. From the regression, the 4th-order polynomial turned out to represent the data adequately:

$$P(x_1) = -365.7643x_1^4 + 915.1596x_1^3 - 894.0699x_1^2 + 556.8470x_1 + 56.2570$$

Differentiation of this equation yields

$$P'(x_1) = -1463.0572x_1^3 + 2745.4788x_1^2 - 1788.1398x_1 + 556.8470$$

The derivative term is represented as

$$\frac{d \ln P}{dx_1} = \frac{1}{P} \frac{dP}{dx_1} = \frac{-1463.0572x_1^3 + 2745.4788x_1^2 - 1788.1398x_1 + 556.8470}{-365.7643x_1^4 + 915.1596x_1^3 - 894.0699x_1^2 + 556.8470x_1 + 56.2570}$$

thus, we have

$$\frac{dy_1}{dx_1} = \frac{y_1(1-y_1)}{y_1 - x_1} \frac{-1463.0572x_1^3 + 2745.4788x_1^2 - 1788.1398x_1 + 556.8470}{-365.7643x_1^4 + 915.1596x_1^3 - 894.0699x_1^2 + 556.8470x_1 + 56.2570}$$

This differential equation is defined by the MATLAB function *bzacfun*.

```
function dy = bzacfun(x,y)
% dy/dx for benzene/acetic acid system
dy = (y*(1-y)/(y-x)) * (-1463.0572*x^3 + 2745.4788*x^2 - 1788.1398*x ...
+ 556.8470)/(-365.7643*x^4 + 915.1596*x^3 - 894.0699*x^2 + ...
556.8470*x + 56.2570);
end
```

The script file *bzacP* performs the data regression and calculates the activity coefficients. This script generates plots of P , vapor-phase composition, and activity coefficients as a function of the liquid-phase composition. The initial conditions are set to $\epsilon_{x_1} = 10^{-5}$ and $y_1(0) = \epsilon_{y_1}$.

```
% bzacP.m: vapor phase composition and activity coefficient
% for benzene/acetic acid system
```

```

x1 = [0.0 0.0069 0.1565 0.3396 0.4666 0.6004 0.7021 0.8286 0.8862 ...
0.9165 0.9561 0.9840 1.0];
P = [57.52 58.2 126.00 175.30 189.50 224.30 236.00 250.20 259.00 ...
261.11 264.45 266.53 271.00];
c = polyfit(x1,P,4) % regression polynomial (4th order)
dc = polyder(c) % differentiation of the polynomial
x10 = 1e-5; xinv = [x10 1]; y10 = x10*dc(end)/c(end);
[x,y] = ode45(@bzacf, xinv, y10); % solve the differential eqn.
P1v = P(end); P2v = P(1); Px = polyval(c,x);
gam1 = y.*Px./x/P1v; gam2 = (1-y).*Px./(1-x)/P2v;
subplot(2,2,1)
x1i = 0:0.01:1; Pv = polyval(c,x1i);
plot(x1i,Pv,x1,P,'o'), xlabel('x_1'), ylabel('P(mmHg)')
legend('Fitting','Data','Location','best')
subplot(2,2,3)
plot(x,y), xlabel('x_1'), ylabel('y_1') % x1-y1 graph
subplot(2,2,4)
plot(x,gam1,x,gam2,'-.'), xlabel('x_1'), ylabel('\gamma')
legend('\gamma_1','\gamma_2','Location','best'), axis([0 1 0 4])

```

The script *bzacP* produces the following results and generates the plots shown in Figure 4.2:

```

>> bzacP
c =
-365.7643 915.1596 -894.0699 556.8470 56.2570
dc =
1.0e+03 *
-1.4631 2.7455 -1.7881 0.5568

```

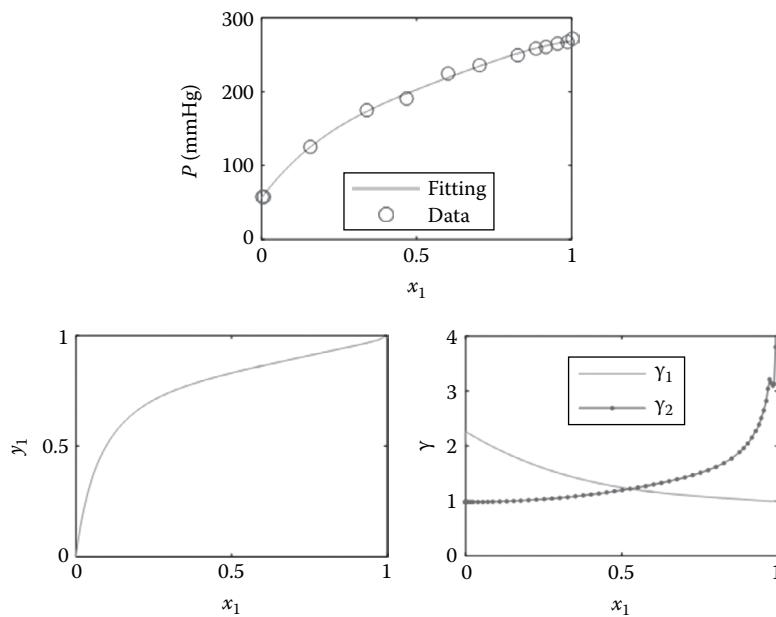


FIGURE 4.2 Total pressure, vapor-phase composition, and activity coefficients for benzene(1)/acetic acid(2) system.

4.5.1 ACTIVITY COEFFICIENT MODELS

4.5.1.1 Wilson Equation

The Wilson equation may be used to estimate activity coefficients for miscible non-ideal systems. For a binary mixture, the activity coefficients γ_1 and γ_2 are obtained from

$$\ln \gamma_1 = -\ln \theta_1 + \frac{(g_{12}\theta_2 - g_{21}\theta_1)x_2}{\theta_1\theta_2}, \quad \ln \gamma_2 = -\ln \theta_2 + \frac{(g_{12}\theta_2 - g_{21}\theta_1)x_1}{\theta_1\theta_2}$$

where

x_i is the mole fraction

$\theta_1 = x_1 + g_{12}x_2$, $\theta_2 = x_2 + g_{21}x_1$

g_{12} and g_{21} can be determined using the experimental data

4.5.1.1.1 Calculation of Wilson Equation Parameters: Use of Azeotropic Data

The parameters of the Wilson equation can be determined by using the azeotropic data. At a low-pressure azeotropic point, the liquid-phase activity coefficients can be obtained from the relation $\gamma_i = P/P_i$.²⁶ At the azeotropic point T_a , the vapor pressure P_i may be estimated by using a simple relation such as the Antoine equation. Thus, the parameters g_{ij} can be determined if the composition data are available. For a binary system, we can get two nonlinear equations:

$$f_1 = \ln \gamma_1 + \ln \theta_1 - \frac{(g_{12}\theta_2 - g_{21}\theta_1)x_2}{\theta_1\theta_2}, \quad f_2 = \ln \gamma_2 + \ln \theta_2 + \frac{(g_{12}\theta_2 - g_{21}\theta_1)x_1}{\theta_1\theta_2}$$

The MATLAB function *wilact* defines these nonlinear equations. This function takes x_1 , x_2 , γ_1 , and γ_2 as inputs.

```
function f = wilact(g,x1,x2,gam1,gam2)
t1 = x1 + g(1)*x2; t2 = x2 + g(2)*x1;
f = [log(gam1) + log(t1) - (g(1)*t2 - g(2)*t1)*x2/(t1*t2);
      log(gam2) + log(t2) + (g(1)*t2 - g(2)*t1)*x1/(t1*t2)];
end
```

4.5.2 VAN LAAR EQUATION

The van Laar equation is frequently used to correlate activity coefficient data for non-ideal systems. The van Laar equations for the correlation of binary activity coefficients are given by²⁷

$$\log \gamma_1 = \frac{Ax_2^2}{\left(\frac{A}{B}x_1 + x_2\right)^2}, \quad \log \gamma_2 = \frac{Bx_1^2}{\left(x_1 + \frac{B}{A}x_2\right)^2}$$

where parameters A and B , constants for a particular binary mixture, may be determined using experimental data.

Example 4.12: Estimation of Parameters of the Wilson Equation

The temperature of the azeotrope for an ethanol(1)/*n*-octane(2) mixture at $P = 760$ mmHg is 77°C and the composition at the azeotrope is 78% ethanol and 22% *n*-octane (% by weight). At the temperature of the azeotrope T , the vapor pressures of ethanol and *n*-octane may be obtained by the

Antoine equation $\log P_i = A_i - (B_i/(T + C_i))$ where T is the azeotrope temperature (°C) and P_i is the vapor pressure (mmHg). For ethanol, $A_1 = 8.04494$, $B_1 = 1554.3$, and $C_1 = 222.65$, and for *n*-octane, $A_2 = 6.92374$, $B_2 = 1355.126$, and $C_2 = 209.517$. The molecular weight of ethanol and *n*-octane are 46.07 and 114, respectively. Determine the Wilson equation coefficients for this system.

Solution

The Wilson equation parameters can be estimated using the vapor pressures obtained by the Antoine equation. A system of nonlinear equations in terms of mole fractions is defined by the MATLAB function *wilact*.

```
function f = wilact(g,x1,x2,gam1,gam2)
t1 = x1 + g(1)*x2; t2 = x2 + g(2)*x1;
f = [log(gam1) + log(t1) - (g(1)*t2 - g(2)*t1)*x2/(t1*t2);
      log(gam2) + log(t2) + (g(1)*t2 - g(2)*t1)*x1/(t1*t2)];
end
```

The script *wilsonpar* uses the built-in function *fsolve* to solve the nonlinear system.

```
% wilsonpar.m: Wilson equation parameters for a binary system
w1 = 78; mw1 = 46.07; mw2 = 114; Ta = 77; P = 760;
A1 = 8.04494; B1 = 1554.3; C1 = 222.65;
A2 = 6.92374; B2 = 1355.126; C2 = 209.517;
% mole fraction
x1 = (w1/mw1)/(w1/mw1 + (100-w1)/mw2); x2 = 1 - x1;
% vapor pressure
P1 = 10^(A1 - B1/(Ta + C1)); P2 = 10^(A2 - B2/(Ta + C2));
% activity coefficients
gam1 = P/P1; gam2 = P/P2;
% solve the nonlinear system (g(1)=g12, g(2)=g21)
g0 = [0.1 0.1]; % initial guess
g = fsolve(@wilact,g0,[],x1,x2,gam1,gam2);
g12 = g(1), g21 = g(2)
```

The script *wilsonpar* produces the following results:

```
>> wilsonpar
g12 =
0.5713
g21 =
0.0998
```

4.5.3 ACTIVITY COEFFICIENTS BY THE GROUP CONTRIBUTION METHOD

A molecule can be considered to be a combination of functional groups. Among the activity models based on the contribution of functional groups, the UNIQUAC (Universal Quasi-Chemical Model) and the UNIFAC (Universal Quasi-Chemical Functional Group Activity Coefficient) methods are widely used.

4.5.3.1 UNIQUAC Method

The activity coefficient can be represented as a sum of a combinatorial term to account for molecular size and shape differences and a residual term to account for molecular interactions²⁸:

$$\ln \gamma_i = \ln \gamma_i^C + \ln \gamma_i^R$$

where

$$\ln \gamma_i^C = 1 - J_i + \ln J_i - 5q_i \left(1 - \frac{J_i}{L_i} + \ln \frac{J_i}{L_i} \right)$$

$$\ln \gamma_i^R = q_i \left(1 - \ln s_i - \sum_j \frac{\theta_j \tau_{ij}}{s_j} \right)$$

$$\tau_{ji} = \exp \left(-\frac{u_{ji} - u_{ii}}{RT} \right), \quad J_i = \frac{r_i}{\sum_j r_j x_j}, \quad L_i = \frac{q_i}{\sum_j q_j x_j}, \quad s_i = \sum_k \theta_k \tau_{ki}, \quad \theta_i = \frac{x_i q_i}{\sum_j x_j q_j}$$

$$r_i = \sum_k v_k^{(i)} R_k, \quad q_i = \sum_k v_k^{(i)} Q_k$$

In these equations, r_i is a parameter representing a relative molecular volume and q_i is a parameter representing a relative molecular surface area, each of which is given by the sum of R_k and Q_k parameters of functional groups comprising the component, respectively. Values of parameters R_k and Q_k can be found elsewhere.^{29,30}

4.5.3.2 UNIFAC Method

The UNIFAC method is based on the UNIQUAC method, and the assumptions regarding coordination numbers, and so forth, are similar to those in the UNIQUAC method. The activity coefficient can be represented as the sum of a combinatorial term and a residual term³¹:

$$\ln \gamma_i = \ln \gamma_i^C + \ln \gamma_i^R$$

where

$$\ln \gamma_i^C = 1 - J_i + \ln J_i - 5q_i \left(1 - \frac{J_i}{L_i} + \ln \frac{J_i}{L_i} \right)$$

$$\ln \gamma_i^R = q_i \left(1 - \sum_k \left(\frac{\theta_k \beta_{ik}}{s_k} - e_{ki} \ln \frac{\beta_{ik}}{s_k} \right) \right)$$

$$e_{ki} = \frac{v_k^{(i)} Q_k}{q_i}, \quad \beta_{ik} = \sum_m e_{mi} \tau_{mk}, \quad \tau_{ij} = \exp \left(-\frac{a_{ij}}{T} \right), \quad J_i = \frac{r_i}{\sum_j r_j x_j}, \quad L_i = \frac{q_i}{\sum_j q_j x_j}, \quad \theta_k = \frac{x_i q_i e_{ki}}{\sum_j x_j q_j}$$

Here, a_{ij} are group interaction parameters and can be found from Reference 32.

The MATLAB function *unifgam* calculates activity coefficients using the UNIFAC method. The basic syntax is

```
gam = unifgam(k, R, Q, nu, amn, Nc, x, T)
```

where k is the number of functional groups; R and Q are the vectors of volumes and surface areas for each functional group, respectively; nu is the number of functional groups contained in each component; amn is the matrix of group interaction parameters; Nc is the number of components; x is the vector of liquid-phase mole fraction; T is the temperature (K); and gam is the vector of activity coefficients for each component.

```
function gam = unifgam(k, R, Q, nu, amn, Nc, x, T)
% Estimation of activity coefficients using the UNIFAC method
% input:
%   R, Q: vectors of volumes and surface areas for each functional group
%   nu: number of functional groups contained in each component
```

```

% (row: functional groups k, column: components i)
% amn: matrix of group interaction parameters
% Nc: number of components
% x: vector of liquid-phase mole fraction
% T: temperature (K)
% output:
% gam: vector of activity coefficients for each component

r = zeros(1,Nc); q = zeros(1,Nc); tau = zeros(k,k); ek = zeros(k,Nc);
theta = zeros(1,k); beta = zeros(Nc,k); s = zeros(1,k); z = 10;
for j = 1:Nc
    r(j) = sum(R.*nu(:,j)'); q(j) = sum(Q.*nu(:,j)'); % row vector(r,q)
end
for i = 1:k
    ek(i,:) = nu(i,:)*Q(i)./q;
end

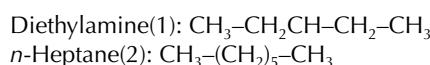
tau = exp(-amn/T);
for i = 1:Nc
    for j = 1:k
        beta(i,j) = sum(ek(:,i).*tau(:,j));
    end
end
for i = 1:k
    theta(i) = sum(x.*q.*ek(i,:))/sum(x.*q);
end

for i = 1:k
    s(i) = sum(theta' .* tau(:,i));
end
J = r/sum(r.*x); L = q/sum(q.*x);
gamc = 1 - J + log(J) - 5*q.* (1 - J./L + log(J./L));
gamr = [];
for i = 1:Nc
    sumb = 0;
    for j = 1:k
        gamval = theta(j)*beta(i,j)/s(j) - ek(j,i)*log(beta(i,j)/s(j));
        sumb = sumb + gamval;
    end
    gamr = [gamr q(i)*(1 - sumb)];
end
gam = exp(gamc + gamr);
end

```

Example 4.13: Activity Coefficients by the UNIFAC Method³³

Determine γ_1 and γ_2 for the binary system diethylamine(1)/*n*-heptane(2) at $T = 308.15$ K when $x_1 = 0.4$ and $x_2 = 0.6$. The subgroups involved are indicated by the chemical formulas as



Solution

We can see that there are three functional groups: CH_3 ($k = 1$), CH_2 ($k = 2$), CH_2NH ($k = 3$). Table 4.11 shows the subgroups, their identification numbers k , values of parameters R_k and Q_k , and $v_j^{(i)}$, which represents the number of j group in component i .²⁹

TABLE 4.11
Parameters of Subgroups

Functional Group	j	R_k	Q_k	$v_j^{(1)}$	$v_j^{(2)}$
CH ₃	1	0.9011	0.8480	2	2
CH ₂	2	0.6744	0.5400	1	5
CH ₂ NH	3	1.2070	0.9360	1	0

Source: Poling, B.E. et al., *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, 2001, pp. 8.78–8.81.

Since CH₃ and CH₂ belong to the main group 1 and CH₂NH belongs to the main group 15, the a_{mn} matrix is given by

		Main Group 1		Main Group 15
		$j = 1$	$j = 2$	$j = 3$
Main Group 1	$j = 1$	0	0	255.7
	$j = 2$	0	0	255.7
Main Group 15		$j = 3$	65.33	0

The script *useunifgam* sets required data and calls the function *unifgam* to perform the UNIFAC calculation.

```
% useunifgam.m
Nc = 2; k = 3; % numbers of components(Nc) and functional groups(k)
% vector of volumes for each functional group
R = [0.9011 0.6744 1.2070];
% vector of surface areas for each functional group
Q = [0.8480 0.5400 0.9360];
% number of functional groups contained in each component
% (row: functional groups k, column: components i)
nu = [2 2; 1 5; 1 0];
% matrix of group interaction parameters
amn = [0 0 255.7; 0 0 255.7; 65.33 65.33 0];
x = [0.4 0.6]; % mole fraction of each component
T = 308.15; % T(K)
gam = unifgam(k,R,Q,nu,amn,Nc,x,T)
```

The script *useunifgam* generates the following results:

```
>> useunifgam
gam =
1.1330 1.0470
```

Example 4.14: Activity Coefficients for a 4-Component Mixture by the UNIFAC Method

Estimate activity coefficients of all component for the system *n*-hexane(1)/ethanol(2)/methylcyclopentane(3)/benzene(4). The given pressure P is 1 atm, the temperature T is 334.82 K, and the liquid-phase mole fraction of each component is $x_1 = 0.162$, $x_2 = 0.068$, $x_3 = 0.656$, and $x_4 = 0.114$. Each component consists of the following functional groups:

n-Hexane(1): 2CH₃–4CH₂
Ethanol(2): CH₃–CH₂–OH

TABLE 4.12
Parameters of Subgroups

Functional Group	j	R_k	Q_k	$v_j^{(1)}$	$v_j^{(2)}$	$v_j^{(3)}$	$v_j^{(4)}$
CH ₃	1	0.9011	0.848	2	1	3	0
CH ₂	2	0.6744	0.540	4	1	1	0
CH	3	0.4469	0.228	0	0	1	0
C	4	0.2195	0.000	0	0	1	0
ACH	5	0.5313	0.400	0	0	0	6
OH	6	1.0000	1.200	0	1	0	0

Source: Poling, B.E. et al., *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, 2001, pp. 8.78–8.81.

TABLE 4.13
Interaction Parameter Matrix for a 4-Component System

	Main Group 1				Main Group 3		Main Group 5	
	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$		
Main Group 1	$j = 1$	0	0	0	0	61.13	986.5	
	$j = 2$	0	0	0	0	61.13	986.5	
	$j = 3$	0	0	0	0	61.13	986.5	
	$j = 4$	0	0	0	0	61.13	986.5	
Main Group 3	$j = 5$	−11.12	−11.12	−11.12	−11.12	0	636.1	
Main Group 5	$j = 6$	156.4	156.4	156.4	156.4	89.60	0	

Source: Poling, B.E. et al., *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, 2001, pp. 8.82–8.93.

Methylcyclopentane(3): 3CH₃−CH₂−CH−C
Benzene(4): 6ACH

Solution

We can see that there are six functional groups: CH₃ ($j = 1$), CH₂ ($j = 2$), CH ($j = 3$), C ($j = 4$), ACH ($j = 5$), and OH ($j = 6$). Table 4.12 shows the subgroups, their identification numbers k , values of parameters R_k and Q_k , and $v_j^{(1)}$, which represents the number of j group in component i .²⁹

CH₃, CH₂, CH, and C belong to the main group 1, ACH belongs to the main group 3, and OH belongs to the main group 5. Thus, we can construct the a_{mn} parameter matrix as shown in Table 4.13.³²

The script *useunifgam2* sets required data and calls the function *unifgam* to perform the UNIFAC calculation.

```
% useunifgam2.m
Nc = 4; k = 6; % numbers of components (Nc) and functional groups (k)
R = [0.9011 0.6744 0.4469 0.2195...
      0.5313 1.0000]; % vector of volumes for each functional group
Q = [0.848 0.540 0.228 0.000 0.400...
      1.200]; % vector of surface areas for each functional group
% number of functional groups contained in each component
% (row: functional groups k, column: components i)
nu = [2 1 3 0; 4 1 1 0; 0 0 1 0; 0 0 1 0;...
      0 0 0 6; 0 1 0 0];
```

```

amn = [0 0 0 0 61.13 986.5; 0 0 0 0 61.13 986.5; 0 0 0 0 61.13 986.5;...
        0 0 0 0 61.13 986.5; -11.12 -11.12 -11.12 -11.12 0 636.1;...
        156.4 156.4 156.4 156.4 89.60 0]; % matrix of group interaction parameters
x = [0.162 0.068 0.656 0.114]; % % mole fraction of each component
T = 334.82; % T(K)
gam = unifgam(k, R, Q, nu, amn, Nc, x, T)

```

The script useunifgam2 generates the following results:

```

>> useunifgam
gam =
1.0463 8.1079 1.0392 1.3088

```

4.6 VAPOR-LIQUID EQUILIBRIUM

4.6.1 VAPOR-LIQUID EQUILIBRIUM BY RAOULT'S LAW

If we assume that the vapor phase is an ideal gas and the liquid phase is an ideal solution, the equilibrium criteria becomes

$$y_i P = x_i P_i^{sat} \quad (i = 1, 2, \dots, N)$$

The ratio of the vapor-phase mole fraction to the liquid-phase mole fraction, K_i , is defined as

$$K_i = \frac{y_i}{x_i} = \frac{P_i^{sat}}{P}$$

K_i can be estimated approximately by the relation³⁴

$$K_i \approx \frac{P_{c,i}}{P} 10^{\frac{7}{3}(1+w)\left(1-\frac{1}{T_{r,i}}\right)}$$

which is sometimes called the shortcut K -ratio.

From the expression to Raoult's law, the vapor-phase mole fraction is given by

$$y_j = \frac{x_j P_j^{sat}}{P}$$

where

x_j is a liquid-phase mole fraction of species j

P is the total pressure

P_j^{sat} is the vapor pressure of pure species j at the temperature of the system

$\sum_j y_j = 1$ and the above equation may be summed over all species to yield

$$\sum_j y_j = 1 = \sum_j K_j x_j = \frac{1}{P} \sum_j x_j P_j^{sat}$$

where $K_j = y_j/x_j = P_j^{sat}/P$. For the dew-point calculation where liquid-phase compositions are not known, we use

$$\sum_j x_j = 1 = \sum_j \frac{y_j}{K_j} = P \sum_j \frac{y_j}{P_j^{sat}}$$

In flash calculations, let F , V , and L be flow rates (lb mol/hr) of multicomponent feed, vapor, and liquid phases, respectively, and let z_j , y_j , and x_j be the mole fraction of species j in feed, vapor, and liquid phases, respectively. Combination and rearrangement of the total mass balance and the component balance $F = V + L$, $Fz_j = Vy_j + Lx_j$ ($j = 1, \dots, N$) yield

$$x_j = \frac{z_j}{1 + \alpha(K_j - 1)}, \quad y_j = K_j x_j = \frac{K_j z_j}{1 + \alpha(K_j - 1)}$$

where N is the number of components and $\alpha = V/F$. Using the relation $\sum_{j=1}^N x_j = \sum_{j=1}^N y_j = 1$ (i.e., $\sum_{j=1}^N (x_j - y_j) = 0$), we have the following nonlinear equation:

$$f(\alpha) = \sum_{j=1}^N (x_j - y_j) = \sum_{j=1}^N \frac{(1 - K_j)z_j}{1 + \alpha(K_j - 1)} = 0$$

Example 4.15: Raoult's Law³⁵

Binary system acetonitrile(1)/nitromethane(2) conforms closely to Raoult's law. Vapor pressures for the pure species are given by the following Antoine equations:

$$\ln P_1 = A_1 - \frac{B_1}{T + C_1} = 14.2724 - \frac{2945.47}{T + 224.0}, \quad \ln P_2 = A_2 - \frac{B_2}{T + C_2} = 14.2043 - \frac{2972.64}{T + 209.0}$$

where T is the temperature (°C).

1. Generate a graph showing the total pressure P versus x_1 and for a temperature of 75°C.
2. Generate a graph showing T versus x_1 and y_1 for a pressure of $P = 70$ kPa.

Solution

1. For a given T ,

$$\begin{aligned} P &= \sum_i x_i P_i^{sat} = x_1 P_1^{sat} + x_2 P_2^{sat} = P_2^{sat} + \left(P_1^{sat} - P_2^{sat} \right) x_1 \\ &= \exp \left(A_2 - \frac{B_2}{T + C_2} \right) + \left\{ \exp \left(A_1 - \frac{B_1}{T + C_1} \right) - \exp \left(A_2 - \frac{B_2}{T + C_2} \right) \right\} x_1 \\ y_1 &= \frac{x_1 P_1^{sat}}{P} = \frac{x_1}{P} \exp \left(A_1 - \frac{B_1}{T + C_1} \right) \end{aligned}$$

The script *bubbp* plots P versus x_1 and y_1 .

```
% bubbp.m: calculation of bubble p
clear all;
T = 75;
A1 = 14.2724; B1 = 2945.47; C1 = 224;
```

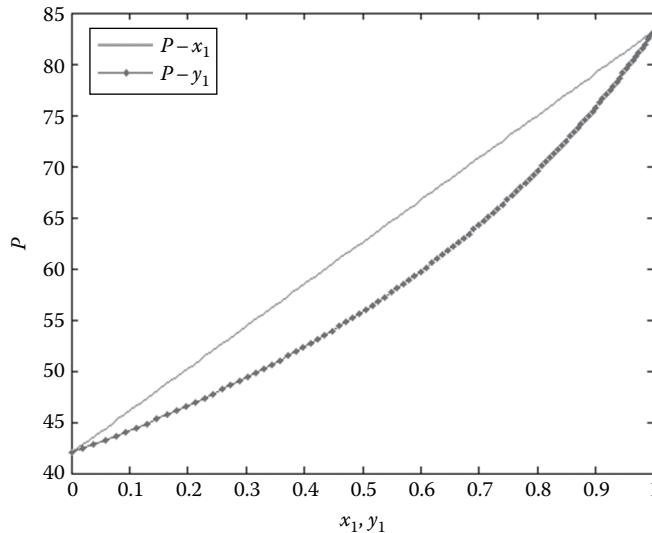


FIGURE 4.3 Vapor pressure of acetonitrile(1)/nitromethane(2) system.

```

A2 = 14.2043; B2 = 2972.64; C2 = 209;
fp = @(x) exp(A2-B2/(T+C2)) + x*(exp(A1-B1/(T+C1)) - exp(A2-B2/(T+C2)));
x1 = 0:0.01:1; y1 = []; P = [];
for k = 1:length(x1)
    Ps = feval(fp,x1(k));
    fy = @(x) x*(exp(A1-B1/(T+C1)))/Ps;
    y = feval(fy,x1(k));
    y1 = [y1 y];
    P = [P Ps];
end
plot(x1,P,y1,P,'.-'), xlabel('x_1, y_1'), ylabel('P')
legend('P-x_1','P-y_1','location','best')

```

Implementation of the script *bubbp* produces [Figure 4.3](#):

```
>> bubbp
```

2. For a given pressure P , T can be found by solving the nonlinear equation

$$\exp\left(A_2 - \frac{B_2}{T+C_2}\right) + \left\{ \exp\left(A_1 - \frac{B_1}{T+C_1}\right) - \exp\left(A_2 - \frac{B_2}{T+C_2}\right) \right\} x_1 - P = 0$$

Once T is known, the vapor-phase composition can be determined from the relation

$$y_1 = \frac{x_1 P_1^{\text{sat}}}{P} = \frac{x_1}{P} \exp\left(A_1 - \frac{B_1}{T+C_1}\right)$$

The script *dewpt* uses the built-in function *fzero* to solve the nonlinear equation.

```

% dewpt.m: calculation of dew point
clear all;
A1 = 14.2724; B1 = 2945.47; C1 = 224;
A2 = 14.2043; B2 = 2972.64; C2 = 209;
T0 = 60; P = 70;
x = 0:0.01:1; T = []; y = [];
for k = 1:length(x)
    x1 = x(k);
    fp = @(T) exp(A2-B2/(T+C2)) + ...
        x1*(exp(A1-B1/(T+C1)) - exp(A2-B2/(T+C2))) - P;
    T = fzero(fp, T0);
    y = [y T];
end

```

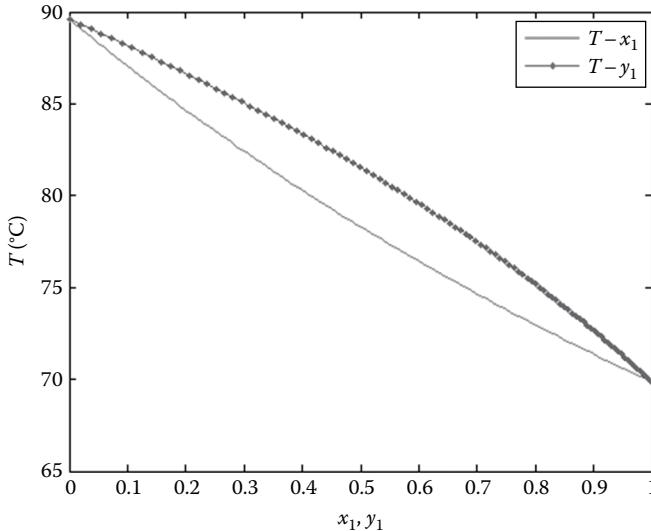


FIGURE 4.4 Dew point of acetonitrile(1)/nitromethane(2) system.

```

T1 = fzero(fp,T0); y1 = x1*exp(A1-B1/(T1+C1))/P;
T = [T T1]; y = [y y1];
end
plot(x,T,y,T,'.-'), xlabel('x_1, y_1'), ylabel('T(°C)')
legend('T-x_1','T-y_1','location','best')

```

Implementation of the script *dewpt* produces a plot showing T versus x_1 and y_1 (see Figure 4.4):

```
>> dewpt
```

4.6.2 VAPOR-LIQUID EQUILIBRIUM BY MODIFIED RAOUlt's LAW

Insertion of an activity coefficient, γ_i , into Raoult's law yields the modified Raoult's law:

$$y_i P = x_i \gamma_i P_i^{sat} \quad (i = 1, 2, \dots, N)$$

The equilibrium ratio (or K -value), K_i , is defined as

$$K_i = \frac{y_i}{x_i} = \frac{\gamma_i P_i^{sat}}{P}$$

According to this equation, $y_i = K_i x_i$ and $x_i = y_i / K_i$. Summation relations give

$$\sum_{i=1}^N y_i = \sum_{i=1}^N K_i x_i = 1, \quad \sum_{i=1}^N x_i = \sum_{i=1}^N \frac{y_i}{K_i} = 1$$

In vapor-liquid equilibrium calculations, the following relations are commonly used:

$$P = \sum_{i=1}^N x_i \gamma_i P_i^{sat}, \quad y_i = \frac{x_i \gamma_i P_i^{sat}}{P} = K_i x_i \quad \text{or} \quad x_i = \frac{y_i P}{\gamma_i P_i^{sat}} = \frac{y_i}{K_i}, \quad \sum_{i=1}^N y_i = 1, \quad \sum_{i=1}^N x_i = 1$$

1. *A bubble-point pressure calculation (Bubble P):* For a given temperature T and liquid-phase compositions x_i , the total pressure P and vapor-phase compositions y_i are calculated. The values of the vapor pressure and the activity coefficient of species i (P_i^{sat} and γ_i) are estimated first. Then P and y_i can be obtained from the relations

$$P = \sum_{i=1}^N x_i \gamma_i P_i^{sat}, \quad y_i = \frac{x_i \gamma_i P_i^{sat}}{P}$$

2. *A dew-point pressure calculation (Dew P):* For a given temperature T and vapor-phase compositions y_i , the total pressure P and liquid-phase compositions x_i are calculated. The vapor pressure of species i (P_i^{sat}) is determined first from the given T followed by the estimation of liquid-phase compositions x_i using the nonlinear equation

$$P = \sum_{i=1}^N x_i \gamma_i P_i^{sat}, \quad f(x) = 1 - \sum_{i=1}^N x_i = 1 - \sum_{i=1}^N \frac{y_i P}{\gamma_i P_i^{sat}} = 0$$

3. *A bubble-point temperature calculation (Bubble T):* For a given total pressure P and liquid-phase compositions x_i , temperature T and vapor-phase compositions y_i are calculated. The solution of the following nonlinear equation yields the value of T : $f(T) = P - \sum_{i=1}^N x_i \gamma_i P_i^{sat} = 0$.

4. *A dew-point temperature calculation (Dew T):* For a given total pressure P and vapor-phase compositions y_i , temperature T and liquid-phase compositions x_i are calculated. We first assume the value of T and calculate the vapor pressure. Then we assume an initial value of the liquid-phase mole fraction x_i (usually Raoult's law is used: $x_i = P y_i / P_i^{sat}$). We calculate the activity coefficient γ_i and solve the nonlinear equation $f(T) = 1 - \sum_{i=1}^N y_i P / \gamma_i P_i^{sat} = 0$ to obtain the value of T . Using the resultant T , the relation $x_i = \frac{y_i P}{\gamma_i P_i^{sat}}$ is evaluated and compared with the initial guess. This procedure is repeated until the value of x_i does not show any changes.

Inputs and outputs for each calculation are summarized in [Table 4.14](#).

Example 4.16: Equilibrium Calculations Using the Modified Raoult's Law³⁶

For the methanol(1)/methyl acetate(2) binary system, reasonable correlations for the activity coefficients are given by

$$\ln \gamma_1 = Ax_2^2, \quad \ln \gamma_2 = Ax_1^2, \quad A = 2.771 - 0.00523T$$

TABLE 4.14
Inputs and Outputs for Vapor–Liquid Equilibrium Calculations

Calculation Type	Inputs	Outputs
Bubble-point pressure (Bubble P)	T, x_i	P, y_i
Dew-point pressure (Dew P)	T, y_i	P, x_i
Bubble-point temperature (Bubble T)	P, x_i	T, y_i
Dew-point temperature (Dew T)	P, y_i	T, x_i

The Antoine equation provides expressions for vapor pressures (kPa):

$$\ln P_1^{\text{sat}} = 16.59158 - \frac{3643.31}{T - 33.324}, \quad \ln P_2^{\text{sat}} = 14.25326 - \frac{2665.54}{T - 53.424}$$

where T is the temperature (K).

1. Estimate P and y_i for $T = 318.15$ K and $x_1 = 0.25$.
2. Estimate P and x_i for $T = 318.15$ K and $y_1 = 0.60$.
3. Estimate T and y_i for $P = 101.33$ kPa and $x_1 = 0.85$.
4. Estimate T and x_i for $P = 101.33$ kPa and $y_1 = 0.40$.

Solution

1. For given temperature and liquid-phase composition, P_i^{sat} and γ_i are calculated. These values are used to determine P and y_i from the relations $P = \sum_{i=1}^N x_i \gamma_i P_i^{\text{sat}}$ and $y_i = x_i \gamma_i P_i^{\text{sat}} / P$.
2. P_i^{sat} is determined from the given temperature and the composition is found by solving the nonlinear equation

$$P = \sum_{i=1}^N x_i \gamma_i P_i^{\text{sat}}, \quad f(x) = 1 - \sum_{i=1}^N x_i = 1 - \sum_{i=1}^N \frac{y_i P}{\gamma_i P_i^{\text{sat}}} = 0$$

The nonlinear equation is defined by the MATLAB function *dewpf*.

```
function f = dewpf(x1,y,T,Psat)
x = [x1 1-x1]; gamma = gamma12(x1,T);
P = sum(x.*gamma.*Psat);
f = sum(y.*P./ (gamma.*Psat)) - 1;
end
```

3. The value of T is obtained from the solution of the nonlinear equation

$$f(T) = P - \sum_{i=1}^N x_i \gamma_i P_i^{\text{sat}} = 0$$

The nonlinear equation is defined by the MATLAB function *bubtf*.

```
function f = bubtf(T,x1,A,B,C,P)
x = [x1 1-x1]; gamma = gamma12(x1,T); Psat = vp12(A,B,C,T);
f = sum(x.*gamma.*Psat) - P;
end
```

4. The vapor pressure is estimated using the assumed T , and the initial value of liquid-phase mole fraction x_i is assumed. Usually Raoult's law is used to assume $x_i = P y_i / P_i^{\text{sat}}$. The activity coefficient γ_i is calculated and T is determined from the nonlinear equation

$$f(T) = 1 - \sum_{i=1}^N \frac{y_i P}{\gamma_i P_i^{\text{sat}}} = 0$$

The nonlinear equation is defined by the MATLAB function *dewpf2*. The resultant T is used to evaluate $x_i = y_i P / \gamma_i P_i^{\text{sat}}$, which is compared with the initial guess. This procedure is repeated until x_i converges on a fixed value.

```
function fp = dewtf2(T,y1,x1,A,B,C,P)
y = [y1 1-y1]; Psat = vp12(A,B,C,T);
gamma = gamma12(x1,T);
fp = sum(y.*P./ (gamma.*Psat)) - 1;
end
```

The script *binVLE* implements these calculation procedures. Using given Antoine equation parameters, this script performs vapor-liquid equilibrium calculations.

```
% binVLE.m: VLE calculation for binary systems using modified Raoult's law
% Antoine vapor pressure equation parameters
A = [16.59158 14.25326]; B = [3643.31 2665.54]; C = [33.424 53.424];

% (1) Bubble P calculation
T = 318.15; x1 = 0.25; x = [x1 1-x1]; % data
gamma = gamma12(x1,T); Psat = vp12(A,B,C,T);
P = sum(x.*gamma.*Psat); y = (x.*gamma.*Psat)/P;
fprintf('(1) Bubble P: P = %g, y1 = %g, y2 = %g\n',P,y(1),y(2));
fprintf('           gammal = %g, gamma2 = %g\n',gamma(1),gamma(2));

% (2) Dew P calculation
T = 318.15; y1 = 0.60; y = [y1 1-y1]; Psat = vp12(A,B,C,T); % data
x10 = 0.7; % initial guess for mole fraction x
[x1, fval] = fzero(@dewpf,x10,[],y,T,Psat); x = [x1 1-x1];
gamma = gamma12(x1,T); P = sum(x.*gamma.*Psat);
fprintf('(2) Dew P: P = %g, x1 = %g, x2 = %g\n',P,x(1),x(2));
fprintf('           gammal = %g, gamma2 = %g\n',gamma(1),gamma(2));

% (3) Bubble T calculation
P = 101.33; x1 = 0.85; x = [x1 1-x1]; % data
T0 = 300; % initial guess for temperature
[T, fval] = fzero(@bubtf,T0,[],x1,A,B,C,P);
gamma = gamma12(x1,T); Psat = vp12(A,B,C,T); y = (x.*gamma.*Psat)/P;
fprintf('(3) Bubble T: T = %g, y1 = %g, y2 = %g\n',T,y(1),y(2));
fprintf('           gammal = %g, gamma2 = %g\n',gamma(1),gamma(2));

% (4) Dew T calculation
P = 101.33; y1 = 0.40; y = [y1 1-y1]; % data
T0 = 330; % initial guess for temperature
Psat0 = vp12(A,B,C,T0); x1 = y1*P/Psat0(1); crx = 1; cx = 1e-6;
while crx > cx
    [T, fval] = fzero(@dewtf2,T0,[],y1,x1,A,B,C,P);
    gamma = gamma12(x1,T); Psat = vp12(A,B,C,T);
    x = y*P./(gamma.*Psat);
    crx = abs(x - x1);
    x1 = x(1);
end
fprintf('(4) Dew T: T = %g, x1 = %g, x2 = %g\n',T,x(1),x(2));
fprintf('           gammal = %g, gamma2 = %g\n',gamma(1),gamma(2));
```

Implementation of the script *binVLE* produces the following outputs:

```
>> binVLE
(1) Bubble P: P = 73.5003, y1 = 0.282205, y2 = 0.717795
           gammal = 1.86401, gamma2 = 1.07164
(2) Dew P: P = 62.8945, x1 = 0.816927, x2 = 0.183073
           gammal = 1.03780, gamma2 = 2.09348
(3) Bubble T: T = 331.201, y1 = 0.66967, y2 = 0.33033
           gammal = 1.02365, gamma2 = 2.11816
(4) Dew T: T = 326.697, x1 = 0.460197, x2 = 0.539803
           gammal = 1.36283, gamma2 = 1.25231
```

The outputs can be summarized as shown in [Table 4.15](#).

TABLE 4.15
Results of VLE Calculations

Problem Type	<i>P</i> (kPa)	<i>T</i> (K)	<i>x</i> ₁	<i>y</i> ₁	γ_1	γ_2
Bubble <i>P</i>	73.5003	<i>318.15</i>	0.25	0.28221	1.86401	1.07164
Dew <i>P</i>	62.8945	<i>318.15</i>	0.81693	0.60	1.03780	2.09348
Bubble <i>T</i>	<i>101.33</i>	331.201	0.85	0.66967	1.02365	2.11816
Dew <i>T</i>	<i>101.33</i>	326.696	0.46019	0.40	1.36283	1.25231

Note: *Italics* are inputs.

4.6.2.1 Flash Calculations

The dew-point and the bubble-point calculation methods can be used in the estimation of the quantities and compositions for the flash evaporation of an ideal multicomponent mixture. Figure 4.5 shows a flash evaporator.

In Figure 4.5, *F*, *V*, and *L* are total feed, vapor, and liquid flow rates, respectively, and *z_j*, *y_j*, and *x_j* are mole fractions of component *j* in the feed, vapor, and liquid streams, respectively. The total number of components is assumed to be *N*. From the overall mass and component balances *F* = *V* + *L*, *Fz_{j = *Vy_j* + *Lx_j* (*j* = 1, ..., *N*) and using *y_{j = *K_jx_j*, we have}*}*

$$x_j = \frac{z_j}{1 + \alpha(K_j - 1)}, \quad y_j = K_j x_j = \frac{K_j z_j}{1 + \alpha(K_j - 1)}$$

where $\alpha = V/F$, $K_j = \gamma_j P_j^{sat}/P$, P_j^{sat} is the vapor pressure of species *j*, and *P* is the total pressure. Substitution of these equations into the summing relations $\sum_{j=1}^N x_j = \sum_{j=1}^N y_j = 1$ (i.e., $\sum_{j=1}^N (x_j - y_j) = 0$) yields the following nonlinear equation:

$$f(\alpha) = \sum_{j=1}^N (x_j - y_j) = \sum_{j=1}^N \frac{(1 - K_j)z_j}{1 + \alpha(K_j - 1)} = 0$$

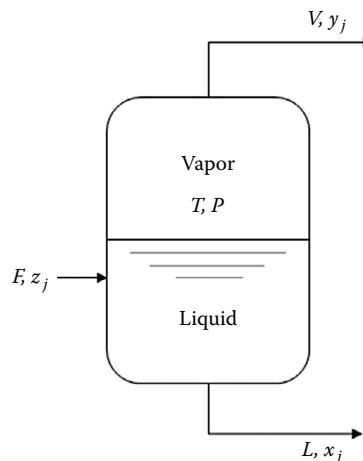


FIGURE 4.5 Flash evaporator.

At dew point, $\alpha = 1$ (all the feed is evaporated) and

$$f(T) = \sum_{j=1}^N \frac{z_j}{K_j} - 1 = 0$$

At bubble point, $\alpha = 0$ (evaporation of the feed is initiated) and we have

$$f(T) = \sum_{j=1}^N z_j K_j - 1 = 0$$

P_j^{sat} is given by the Antoine vapor pressure equation:

$$\log P_j = A_j - \frac{B_j}{C_j + T} \quad (P_j^{sat}: \text{mmHg}, T: {}^\circ\text{C})$$

Example 4.17: Flash Evaporator³⁷

A feed stream of an ideal 4-component mixture is fed into a flash evaporator. The composition of the feed stream is given in Table 4.16 with the Antoine equation constants. The flash drum operates under high pressure, between 15 and 25 atm, with a feed stream at 50°C. Estimate the percent of the total feed at 50°C that is evaporated, $\alpha (=V/F)$, and the corresponding mole fractions in the liquid and vapor streams for $P = 16, 18, 20$, and 24 atm. Calculate the dew-point and the bubble-point temperature of the feed stream.

Solution

The MATLAB function *falp* defines summing relations. The equation for dew-point temperature $f(T) = \sum_{j=1}^N (z_j/K_j) - 1 = 0$ and the equation for bubble-point temperature $f(T) = \sum_{j=1}^N z_j K_j - 1 = 0$ are defined by MATLAB functions *falp1* and *falp0*, respectively.

```
falp.m
function fv = falp(alp,g)
zf = g.zf; k = g.k; fv = sum(zf.* (1-k)./(1 + alp*(k-1)));
end

falp1.m
function fv = falp1(T,g)
% dew-point calculation: alpha=1
z = g.zf; A = g.A; B = g.B; C = g.C; P = g.Pi;
k = 10.^ (A - B. / (C + T)) / P; fv = sum(z./k) - 1;
end
```

TABLE 4.16
Feed Composition and Antoine Equation Constants

Component	Mole Fraction (z_j)	Antoine Equation Constants		
		A	B	C
Ethylene	0.10	6.64380	395.74	266.681
Ethane	0.25	6.82915	663.72	256.681
Propane	0.50	6.80338	804.00	247.040
<i>n</i> -Butane	0.15	6.80776	935.77	238.789

```

falp0.m
function fv = falp0(T,g)
% bubble-point calculation: alpha=0
z = g.zf; A = g.A; B = g.B; C = g.C; P = g.Pi;
k = 10.^ (A - B./(C + T)) / P; fv = sum(z.*k) - 1;
end

The fields of the structure variable g consist of feed compositions and the Antoine equation
constants. The script flashdrum calls falp, falp0, and falp1 and uses the built-in function fzero to
solve the nonlinear equations.

% flashdrum.m: flash calculation
% feed composition and Antoine constants
zf = [0.1 0.25 0.5 0.15];
A = [6.64380 6.82915 6.80338 6.80776];
B = [395.74 663.72 804.00 935.77];
C = [266.681 256.681 247.040 238.789];
P = [16 18 20 24]*760; % pressure (mmHg)

% compute alpha
T = 50; % deg.C
Pv = 10.^ (A - B./(C + T)); % vapor pressure (mmHg)
x = []; y = []; alpha = [];
g.zf = zf; g.Pv = Pv; g.A = A; g.B = B; g.C = C;
for i = 1:length(P)
    k = Pv/P(i); g.k = k; g.Pi = P(i);
    ax = fzero(@falp,0.5,[],g); xv = zf./(1 + ax*(k - 1));
    yv = k.*xv; x = [x xv']; y = [y yv']; alpha = [alpha ax];
end
Pmmhg = P/760;
Pmmhg, x, y, alpha

% dew-point: alpha=1
Tdew = [];
for i = 1:length(P)
    g.Pi = P(i); T = fzero(@falp1,50,[],g); Tdew = [Tdew T];
end
Tdew

% bubble-point: alpha=0
Tbub = [];
for i = 1:length(P)
    g.Pi = P(i); T = fzero(@falp0,50,[],g); Tbub = [Tbub T];
end
Tbub

```

The script *flashdrum* generates the following outputs:

```

>> flashdrum
Pmmhg =
    16      18      20      24
x =
    0.0052    0.0067    0.0086    0.0136
    0.0685    0.0851    0.1032    0.1409
    0.4874    0.5380    0.5708    0.5945
    0.4389    0.3702    0.3174    0.2510
Y =
    0.1055    0.1212    0.1398    0.1848
    0.2605    0.2875    0.3139    0.3571
    0.5007    0.4914    0.4692    0.4072
    0.1333    0.1000    0.0771    0.0508
alpha =
    0.9454    0.8148    0.6967    0.5045

```

TABLE 4.17
Results of Flash Calculations

Component	P (atm)			
	16	18	20	24
$\alpha = V/F$	0.9454	0.8148	0.6967	0.5045
Dew point T (°C)	52.1955	57.5182	62.4550	71.4161
Bubble point T (°C)	-29.4195	-23.3346	-17.6971	-7.4850
x_j	Ethylene	0.0052	0.0067	0.0086
	Ethane	0.0685	0.0851	0.1032
	Propane	0.4874	0.5380	0.5708
	<i>n</i> -Butane	0.4389	0.3702	0.3174
y_j	Ethylene	0.1055	0.1212	0.1398
	Ethane	0.2605	0.2875	0.3139
	Propane	0.5007	0.4914	0.4692
	<i>n</i> -Butane	0.1333	0.1000	0.0771

$$T_{\text{dew}} =$$

$$52.1955 \quad 57.5182 \quad 62.4550 \quad 71.4161$$

$$T_{\text{bub}} =$$

$$-29.4195 \quad -23.3346 \quad -17.6971 \quad -7.4850$$

The results may be summarized as shown in Table 4.17.

4.6.3 VAPOR–LIQUID EQUILIBRIUM USING RATIO OF FUGACITY COEFFICIENTS

4.6.3.1 Dew-Point and Bubble-Point Calculations³⁸

Introduction of the vapor-phase fugacity coefficient to the modified Raoult's law yields, at equilibrium,

$$y_i \hat{\phi}_i P = x_i \gamma_i f_i = x_i \gamma_i \hat{\phi}_i^{\text{sat}} P_i^{\text{sat}}$$

This can be rewritten as

$$y_i \frac{\hat{\phi}_i}{\hat{\phi}_i^{\text{sat}}} P = y_i \Phi_i P = x_i \gamma_i P_i^{\text{sat}}$$

where $\Phi_i = \hat{\phi}_i / \hat{\phi}_i^{\text{sat}}$ is the ratio of fugacity coefficients. The application of virial expansion gives

$$\Phi_i = \exp \left[\left[B_{ii} (P - P_i^{\text{sat}}) + \frac{1}{2} P \sum_j \sum_k y_j y_k (2\delta_{ji} - \delta_{jk}) \right] \frac{1}{RT} \right]$$

$$\delta_{ji} = 2B_{ji} - B_{jj} - B_{ii}, \quad \delta_{jk} = 2B_{jk} - B_{jj} - B_{kk}, \quad \delta_{ii} = \delta_{jj} = 0, \quad \delta_{ij} = \delta_{ji}$$

In dew-point and bubble-point calculations, the following relations are used:

$$y_i = \frac{x_i \gamma_i P_i^{\text{sat}}}{\Phi_i P}, \quad x_i = \frac{y_i \Phi_i P}{\gamma_i P_i^{\text{sat}}}, \quad \sum_i y_i = \sum_i x_i = 1, \quad P = \sum_i \frac{x_i \gamma_i P_i^{\text{sat}}}{\Phi_i}, \quad P = \frac{1}{\sum_i \left(\frac{y_i \Phi_i}{\gamma_i P_i^{\text{sat}}} \right)}$$

1. Bubble-point pressure calculations (Bubble P). Total pressure P and vapor-phase composition y_i are calculated for the given temperature T and liquid-phase composition x_i .
 - a. Compute P_i^{sat} and γ_i , set $\Phi_i = 1$, and calculate $P_{ol} = \sum_i \frac{x_i \gamma_i P_i^{sat}}{\Phi_i}$.
 - b. Determine $y_i = \frac{x_i \gamma_i P_i^{sat}}{\Phi_i P_{ol}}$ and Φ_i .
 - c. Find $P_{new} = \sum_i \frac{x_i \gamma_i P_i^{sat}}{\Phi_i}$. If the criterion $|P_{old} - P_{new}| < \epsilon$ is not satisfied, set $P_{old} = P_{new}$ and go to b.
2. Dew-point pressure calculations (Dew P). For the given values of T and y_i , P and x_i are estimated.
 - a. Calculate P_i^{sat} , set $\Phi_i = 1$ and $\gamma_i = 1$, and compute $P_{old} = \frac{1}{\sum_i (y_i \Phi_i / \gamma_i P_i^{sat})}$. Find $x_i = \frac{y_i \Phi_i P_{old}}{\gamma_i P_i^{sat}}$ and γ_i , and estimate P_{old} again.
 - b. Determine Φ_i .
 - c. Calculate $x_i = \frac{y_i \Phi_i P_{old}}{\gamma_i P_i^{sat}}$, normalize x_i , and find γ_i . Repeat this procedure until the change in γ_i from one iteration to the next is less than some tolerance.
 - d. Calculate $P_{new} = \frac{1}{\sum_i (y_i \Phi_i / \gamma_i P_i^{sat})}$. If the criterion $|P_{old} - P_{new}| < \epsilon$ is not satisfied, set $P_{old} = P_{new}$ and go to b.
3. Bubble-point temperature calculations (Bubble T). For the given values of P and x_i , T and y_i are estimated.
 - a. Set $\Phi_i = 1$, and compute $T_i^{sat} = \frac{B_i}{A_i - \ln P} - C_i$ and $T = \sum_i x_i T_i^{sat}$.
 - b. Calculate P_i^{sat} and γ_i . For a component j , calculate $P_j^{sat} = \frac{P}{\sum_i (x_i \gamma_i / \Phi_i) (P_i^{sat} / P_j^{sat})}$ and $T = \frac{B_j}{A_j - \ln P_j^{sat}} - C_j$.
 - c. Determine P_i^{sat} and find $y_i = \frac{x_i \gamma_i P_i^{sat}}{\Phi_i P}$. Calculate Φ_i and γ_i , and compute $P_j^{sat} = \frac{P}{\sum_i (x_i \gamma_i / \Phi_i) (P_i^{sat} / P_j^{sat})}$ and $T = \frac{B_j}{A_j - \ln P_j^{sat}} - C_j$. Repeat this procedure until the change in T from one iteration to the next is less than some tolerance.
4. Dew-point temperature calculations (Dew T). For the given values of P and y_i , T and x_i are estimated.
 - a. Set $\Phi_i = 1$ and $\gamma_i = 1$, and compute $T_i^{sat} = \frac{B_i}{A_i - \ln P} - C_i$ and $T = \sum_i x_i T_i^{sat}$.
 - b. Find P_i^{sat} . For a component j , calculate $P_j^{sat} = P \sum_i \frac{y_i \Phi_i}{\gamma_i} (P_j^{sat} / P_i^{sat})$ and $T = \frac{B_j}{A_j - \ln P_j^{sat}} - C_j$.
 - c. Determine P_i^{sat} and Φ_i , and calculate $x_i = \frac{y_i \Phi_i P}{\gamma_i P_i^{sat}}$ and γ_i .
 - d. Calculate $P_j^{sat} = P \sum_i \frac{y_i \Phi_i}{\gamma_i} (P_j^{sat} / P_i^{sat})$ and $T = \frac{B_j}{A_j - \ln P_j^{sat}} - C_j$.
 - e. Find P_i^{sat} and Φ_i .

- f. Calculate $x_i = \frac{y_i \Phi_i P}{\gamma_i P_i^{sat}}$ followed by normalization, and determine γ_i . Repeat this procedure until the change in γ_i from one iteration to the next is less than some tolerance.
- g. Calculate $P_j^{sat} = P \sum_i \frac{y_i \Phi_i}{\gamma_i} \left(P_j^{sat} / P_i^{sat} \right)$ and $T_{new} = \frac{B_j}{A_j - \ln P_j^{sat}} - C_j$. If the criterion $|T - T_{new}| < \epsilon$ is not satisfied, set $T = T_{new}$ and go to e.

Example 4.18: Bubble T Calculations for a 4-Component System³⁹

Determine the bubble temperature and vapor-phase mole fractions for *n*-hexane(1)/ethanol(2)/methylcyclopentane(3)/benzene(4) system. The given pressure is 1 atm, and the given liquid-phase mole fractions are $x_1 = 0.162$, $x_2 = 0.068$, $x_3 = 0.656$, and $x_4 = 0.114$. Vapor pressure can be estimated using the Antoine equation $\ln P_i^{sat} = A_i - (B_i / (T + C_i))$ (T : K, P : atm). The parameters for the Antoine equation are

$$A_1 = 9.2033, \quad A_2 = 12.2786, \quad A_3 = 9.1690, \quad A_4 = 9.2675$$

$$B_1 = 2697.55, \quad B_2 = 3803.98, \quad B_3 = 2731.00, \quad B_4 = 2788.51$$

$$C_1 = -48.78, \quad C_2 = -41.68, \quad C_3 = -47.11, \quad C_4 = -52.36$$

The virial coefficients (cm³/mol) for each component are

$$B_{11} = -1360.1, \quad B_{12} = -657.0, \quad B_{13} = -1274.2, \quad B_{14} = -1218.8$$

$$B_{22} = -1174.7, \quad B_{23} = -621.8, \quad B_{24} = -589.7, \quad B_{33} = -1191.9$$

$$B_{34} = -1137.9, \quad B_{44} = -1086.9$$

Solution

This problem belongs to the bubble-point temperature calculations. Thus, the computational procedure begins by setting $\Phi_i = 1$ and calculating $T_i^{sat} = (B_i / (A_i - \ln P)) - C_i$ and $T = \sum_i x_i T_i^{sat}$. The gas constant $R = 82.06 \text{ cm}^3 \cdot \text{atm}/(\text{mol} \cdot \text{K})$ is used in the calculation of Φ_i . For the calculation of activity coefficients, the UNIFAC method is employed. The script *bubbleT* implements each computational step for bubble-point temperature calculations. This script calls the function *unifgam* to estimate activity coefficients.

```
% bubbleT.m : bubble temperature of a mixture
clear all;

% Data
n = 4; % number of components
P = 1; % total pressure (atm)
Rg = 82.06; % gas constant (cm^3 atm/(mol K))
x = [0.162 0.068 0.656 0.114]; % liquid-phase mole fraction
% Antoine equation constants
A = [9.2033 12.2786 9.1690 9.2675];
B = [2697.55 3803.98 2731.00 2788.51];
C = [-48.78 -41.68 -47.11 -52.36];
% Virial coefficients
Bij = zeros(n,n);
Bij(1,1) = -1360.1; Bij(1,2) = -657.0; Bij(1,3) = -1274.2; Bij(1,4) = -1218.8;
Bij(2,2) = -1174.7; Bij(2,3) = -621.8; Bij(2,4) = -589.7;
Bij(3,3) = -1191.9; Bij(3,4) = -1137.9;
Bij(4,4) = -1086.9;
```

```

% Data for calculation of activity coefficients using UNIFAC method
k = 6; % number of functional groups (k)
R = [0.9011 0.6744 0.4469 0.2195...
      0.5313 1.0000]; % volume vector for each functional group
Q = [0.848 0.540 0.228 0.000 0.400...
      1.200]; % surface area vector for each functional group
nu = [2 1 3 0; 4 1 1 0; 0 0 1 0; 0 0 1 0;...
      0 0 0 6; 0 1 0 0]; % number of functional groups
amn = [0 0 0 0 61.13 986.5; 0 0 0 0 61.13 986.5; 0 0 0 0 61.13 986.5;...
      0 0 0 0 61.13 986.5; -11.12 -11.12 -11.12 -11.12 0 636.1;...
      156.4 156.4 156.4 156.4 89.60 0]; % interaction parameter matrix

% Initialization
Terr = 1; Tcrit = 1e-6;

% delta(i,j)
for i = 1:n-1
    Bij(:,i) = Bij(i,:);
end
delta = zeros(n,n);
for i = 1:n
    for j = 1:n
        delta(j,i) = 2*Bij(j,i) - Bij(i,i) - Bij(j,j);
        if j == i
            delta(i,j) = 0;
        end
    end
end
for i = 1:n-1
    delta(:,i) = delta(i,:);
end

% Step 1)
PHIi = ones(1,n); % PHI_i = 1 for each component
Tsat = B./(A - log(P)) - C; % Ti^sat
T = sum(x.*Tsat);

% Step 2)
Psat = exp(A - B./(T + C)); % Pi^sat
gamma = unifgam(k,R,Q,nu,amn,n,x,T);
P1sat = P/(sum(x.*gamma.*Psat./PHIi./Psat(1))); % j=1
T = B(1)/(A(1) - log(P1sat)) - C(1);

% Step 3)
Told = T;
while Terr > Tcrit
    Psat = exp(A - B./(Told + C)); % Pi^sat
    y = x.*gamma.*Psat./PHIi./P;
    for i = 1:n % PHI
        sumy = 0;
        for j = 1:n
            for ki = 1:n
                sumy = sumy + y(j).*y(ki)*(2*delta(j,i) - delta(j,ki));
            end
        end
        PHIi(i) = exp((Bij(i,i)*(P - Psat(i)) + sumy*P/2) / (Rg*Told));
    end
    gamma = unifgam(k,R,Q,nu,amn,n,x,Told);
    P1sat = P/(sum(x.*gamma.*Psat./PHIi./Psat(1))); % j=1
    T = B(1)/(A(1) - log(P1sat)) - C(1);
    Terr = abs(Told - T);
    Told = T;
end
T, y, Psat, PHIi, gamma

```

TABLE 4.18
Results of Bubble T Calculations

Component	x_i (Data)	y_i (Exp) ⁴⁰	Results of Calculations			
			y_i	P_i^{sat}	Φ_i	γ_i
<i>n</i> -Hexane(1)	0.162	0.140	0.1357	0.8053	1.0059	1.0462
Ethanol(2)	0.068	0.274	0.2765	0.5048	1.0052	8.0957
Methylcyclopentane(3)	0.656	0.503	0.5040	0.7317	0.9897	1.0391
Benzene(4)	0.114	0.083	0.0838	0.5525	0.9834	1.3084
T (K)			Experimental: 334.85 K, Calculated: 335.1492 K			

The script *bubbleT* generates the following outputs:

```
>> bubbleT
T =
335.1492
Y =
0.1357 0.2765 0.5040 0.0838
Psat =
0.8053 0.5048 0.7317 0.5525
PHIi =
1.0059 1.0052 0.9897 0.9834
gamma =
1.0462 8.0957 1.0391 1.3084
```

The results can be summarized as shown in Table 4.18.

4.6.3.2 Flash Calculations

Flash calculations based on the ratio of fugacity coefficients are somewhat more complex than calculations based on Raoult's law and K -value correlation, but the primary equations are unchanged. Let N be the number of components; F , V , and L be the flow rates of multicomponent feed, vapor phase, and liquid phase, respectively; and z_j , y_j , and x_j be the mole fractions of species j in the feed, vapor, and liquid, respectively. The overall mass balance and component balance $F = V + L$, $Fz_j = Vy_j + Lx_j$ ($j = 1, \dots, N$) coupled with the relation $y_j = K_j x_j$ yields

$$x_j = \frac{z_j}{1 + \alpha(K_j - 1)}, \quad y_j = K_j x_j = \frac{K_j z_j}{1 + \alpha(K_j - 1)}$$

where $\alpha = V/F$, $K_j = y_j/x_j = \gamma_j P_j^{sat}/\Phi_j P$ ($j = 1, 2, \dots, N$), P_j^{sat} is the vapor pressure of species j , and P is the total pressure. Since both sets of mole fractions sum to unity, $\sum_{j=1}^N x_j = \sum_{j=1}^N y_j = 1$. Thus, if we sum the above equations over all species and subtract unity from these sums, the difference f for each case is zero and we have following nonlinear equations:

$$f(x) = \sum_{j=1}^N x_j - 1 = \sum_{j=1}^N \frac{z_j}{1 + \alpha(K_j - 1)} - 1 = 0$$

$$f(y) = \sum_{j=1}^N y_j - 1 = \sum_{j=1}^N \frac{K_j z_j}{1 + \alpha(K_j - 1)} - 1 = 0$$

$$f(\alpha) = \sum_{j=1}^N (x_j - y_j) = \sum_{j=1}^N \frac{(1-K_j)z_j}{1+\alpha(K_j-1)} = 0$$

At dew point, $\alpha = 1$ and we have

$$f(T) = \sum_{j=1}^N \frac{z_j}{K_j} - 1 = 0$$

Similarly, at bubble point, $\alpha = 0$ and the corresponding nonlinear relation becomes

$$f(T) = \sum_{j=1}^N z_j K_j - 1 = 0$$

The procedure for flash calculations may be summarized as follows:

1. Input temperature, pressure, feed compositions (z_j), and constants. For DEW P calculations, set $y_j = z_j$, and for BUBL P calculations, set $x_j = z_j$ and check $P_{dew} < P < P_{bubble}$. If P is out of this range, stop the calculation procedure. Initialize values of γ_j , Φ_j , α .
2. Calculate $K_j = \gamma_j P_j^{sat}/\Phi_j P$ and find new α from $f(\alpha) = \sum_{j=1}^N \frac{(1-K_j)z_j}{1+\alpha(K_j-1)} = 0$.
3. Calculate $x_j = \frac{z_j}{1+\alpha(K_j-1)}$, $y_j = K_j x_j$, γ_j , and Φ_j . Repeat steps 2 and 3 until the changes in α , x_j and y_j from one iteration to the next are less than some tolerances.

Example 4.19: Flash Calculations for a 4-Component System⁴¹

Perform flash calculations for a 4-component system consisting of *n*-hexane(1)/ethanol(2)/methylcyclopentane(3)/benzene(4). The given pressure and temperature are 1 atm and 334.15 K, and the given mole fractions of the feed stream are $z_1 = 0.25$, $z_2 = 0.40$, $z_3 = 0.20$, and $z_4 = 0.15$. Vapor pressure can be estimated using the Antoine equation $\ln P_i^{sat} = A_i - (B_i / (T + C_i))$ (T : K, P : atm). The parameters for the Antoine equation are

$$A_1 = 9.2033, \quad A_2 = 12.2786, \quad A_3 = 9.1690, \quad A_4 = 9.2675$$

$$B_1 = 2697.55, \quad B_2 = 3803.98, \quad B_3 = 2731.00, \quad B_4 = 2788.51$$

$$C_1 = -48.78, \quad C_2 = -41.68, \quad C_3 = -47.11, \quad C_4 = -52.36$$

The virial coefficients (cm³/mol) for each component are

$$B_{11} = -1360.1, \quad B_{12} = -657.0, \quad B_{13} = -1274.2, \quad B_{14} = -1218.8$$

$$B_{22} = -1174.7, \quad B_{23} = -621.8, \quad B_{24} = -589.7, \quad B_{33} = -1191.9$$

$$B_{34} = -1137.9, \quad B_{44} = -1086.9$$

Solution

The UNIFAC method is used to calculate activity coefficients. The built-in function *fzero* is used to solve the nonlinear equation to find α . The script *flashBubP* implements each computational step for flash calculations (bubble-point pressure calculations). This script calls the function *unifgam* to estimate activity coefficients.

```
% flashBubP.m : flash calculations for multicomponent system (Bubble P)
clear all;

% Data
n = 4; % number of components
P = 1; T = 334.15; % P(atm), T(K)
Rg = 82.06; % gas constant (cm^3 atm/(mol K))
z = [0.25 0.40 0.20 0.15]; % feed stream mole fraction
% Antoine equation constants
A = [9.2033 12.2786 9.1690 9.2675];
B = [2697.55 3803.98 2731.00 2788.51];
C = [-48.78 -41.68 -47.11 -52.36];
% Virial coefficients
Bij = zeros(n,n);
Bij(1,1) = -1360.1; Bij(1,2) = -657.0; Bij(1,3) = -1274.2; Bij(1,4) = -1218.8;
Bij(2,2) = -1174.7; Bij(2,3) = -621.8; Bij(2,4) = -589.7;
Bij(3,3) = -1191.9; Bij(3,4) = -1137.9;
Bij(4,4) = -1086.9;

% Data for calculation of activity coefficients using UNIFAC method
k = 6; % number of functional groups (k)
R = [0.9011 0.6744 0.4469 0.2195...
      0.5313 1.0000]; % volume vector for each functional group
Q = [0.848 0.540 0.228 0.000 0.400...
      1.200]; % surface area vector for each functional group
nu = [2 1 3 0; 4 1 1 0; 0 0 1 0; 0 0 1 0;...
      0 0 0 6; 0 1 0 0]; % number of functional groups
amn = [0 0 0 0 61.13 986.5; 0 0 0 0 61.13 986.5; 0 0 0 0 61.13 986.5;...
      0 0 0 0 61.13 986.5; -11.12 -11.12 -11.12 -11.12 0 636.1;...
      156.4 156.4 156.4 156.4 89.60 0]; % interaction parameter matrix
% Initialization
Aerr = 1; Acrit = 1e-6;

% delta(i,j)
for i = 1:n-1
    Bij(:,i) = Bij(i,:);
end
delta = zeros(n,n);
for i = 1:n
    for j = 1:n
        delta(j,i) = 2*Bij(j,i) - Bij(i,i) - Bij(j,j);
        if j == i
            delta(i,j) = 0;
        end
    end
end
for i = 1:n-1
    delta(:,i) = delta(i,:);
end

% Step 1)
x = z; % Bubble P
Psat = exp(A - B./(T + C)); % Pi^sat
gamma = unifgam(k,R,Q,nu,amn,n,x,T);
PHIi = ones(1,n); % initial guess
Y = x.*gamma.*Psat./PHIi/P;
for i = 1:n % PHI
    sumy = 0;
    for j = 1:n
        for m = 1:n

```

```

        sumy = sumy + y(j)*y(m)*(2*delta(j,i) - delta(j,m));
    end
end
PHII(i) = exp((Bij(i,i)*(P - Psat(i)) + sumy*P/2) / (Rg*T));
end

% Step 2)
K = gamma.*Psat./(PHII*P);
alpha0 = 0.5; alphaold = alpha0;

% Step 3)
while Aerr > Acrit
    falpha = @(alpha) sum((1-K).*z./(1 + alpha*(K - 1)));
    alpha = fzero(falpha, alphaold);
    x = z./(1 + alpha*(K - 1)); y = K.*x;
    gamma = unifgam(k,R,Q,nu,amn,n,x,T);
    for i = 1:n % PHI
        sumy = 0;
        for j = 1:n
            for m = 1:n
                sumy = sumy + y(j)*y(m)*(2*delta(j,i) - delta(j,m));
            end
        end
        PHII(i) = exp((Bij(i,i)*(P - Psat(i)) + sumy*P/2) / (Rg*T));
    end
    K = gamma.*Psat./(PHII*P);
    Aerr = abs(alpha - alphaold);
    alphaold = alpha;
end
alpha, x, y, K, gamma

```

Implementation of the script *flashBubP* produces the following results:

```

>> flashBubP
alpha =
    0.8363
x =
    0.1496 0.5770 0.1316 0.1418
y =
    0.2697 0.3654 0.2134 0.1516
K =
    1.8028 0.6332 1.6212 1.0690
gamma =
    2.3332 1.3150 2.2622 1.9669

```

The results can be summarized as shown in [Table 4.19](#).

TABLE 4.19
Results of Flash Calculations

Component	z_i (Given)	Results of Calculations			
		x_i	y_i	K_i	γ_i
<i>n</i> -Hexane(1)	0.25	0.1496	0.2697	1.8028	2.3332
Ethanol(2)	0.40	0.5770	0.3654	0.6332	1.3150
Methylcyclopentane(3)	0.20	0.1316	0.2134	1.6212	2.2622
Benzene(4)	0.15	0.1418	0.1516	1.0690	1.9669

Note: $T: 334.15 \text{ K}$, $P: 1 \text{ atm}$, calculated value of $\alpha = \frac{V}{F}$: 0.8363.

4.7 VAPOR-LIQUID-LIQUID EQUILIBRIUM

As a mixture of water (80 mol%) and isobutanol (20 mol%) is heated at a constant pressure of 1 atm, two liquid phases coexist. One of these liquid phases is composed principally of water and the other one is composed principally of isobutanol. When the bubble-point temperature is reached at 88.54°C, the two liquid phases are in equilibrium with the vapor phase.⁴² When the temperature is raised above the bubble point, only a single liquid phase and the vapor phase are present until the dew-point temperature is reached. Figure 4.6 shows the phase separation curve for the water-isobutanol system.

Figure 4.7 shows the three-phase flash drum. The temperature and pressure in the flash drum are kept constant at T and P , respectively, and the number of components in the system is n_c . From the overall mass balance, component mass balances, phase equilibrium relations, and summing relations, we have the following equations:

$$\text{Overall mass balance: } F = L_1 + L_2 + V$$

$$\text{Component mass balance: } Fz_i = L_1x_{i,1} + L_2x_{i,2} + Vy_i$$

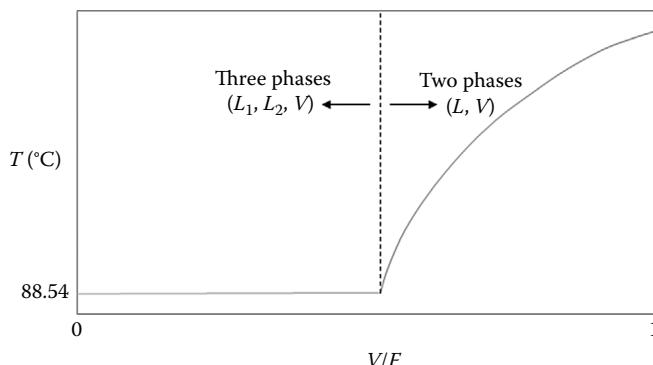


FIGURE 4.6 Phase separation curve for the water-isobutanol system ($P = 1$ atm). (Modified from Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 528.)

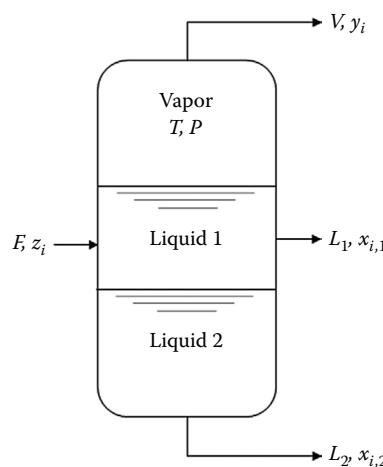


FIGURE 4.7 Three-phase flash drum. (Modified from Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 528.)

Phase equilibrium relations: $y_i = K_{i,1}x_{i,1}$, $y_i = K_{i,2}x_{i,2}$, $K_{i,j} = \frac{\gamma_{i,j}P_i}{P}$ (Raoult's law is used. $\gamma_{i,j}$: activity coefficient of species i in phase j , P : vapor pressure of species i)

Sums of mole fractions: $\sum_{i=1}^{n_c} y_i = 1$, $\sum_{i=1}^{n_c} x_{i,1} = 1$, $\sum_{i=1}^{n_c} x_{i,2} = 1$

$$\Rightarrow \sum_{i=1}^{n_c} y_i - \sum_{i=1}^{n_c} x_{i,1} = 0, \quad \sum_{i=1}^{n_c} x_{i,1} - \sum_{i=1}^{n_c} x_{i,2} = 0$$

From the overall mass balance, we can see that $L_1 = \xi(F - V)$ and $L_2 = (1 - \xi)(F - V)$ where $\alpha = V/F$ and $\xi = L_1/(L_1 + L_2)$ ($0 \leq \alpha, \xi \leq 1$). Substitution of these equations into the component mass balance yields

$$z_i = \left[\frac{\xi(1-\alpha)}{K_{i,1}} + \frac{(1-\xi)(1-\alpha)}{K_{i,2}} + \alpha \right] y_i \quad \text{or} \quad y_i = \left[\frac{z_i}{\frac{\xi(1-\alpha)}{K_{i,1}} + \frac{(1-\xi)(1-\alpha)}{K_{i,2}} + \alpha} \right]$$

Then, from the summing relations, we have

$$\sum_{i=1}^{n_c} y_i - \sum_{i=1}^{n_c} x_{i,1} = \sum_{i=1}^{n_c} \frac{(K_{i,1} - 1)z_i}{\xi(1-\alpha) + (1-\xi)(1-\alpha)\frac{K_{i,1}}{K_{i,2}} + K_{i,1}\alpha} = 0$$

and

$$\sum_{i=1}^{n_c} x_{i,1} - \sum_{i=1}^{n_c} x_{i,2} = \sum_{i=1}^{n_c} \frac{(K_{i,1}/K_{i,2} - 1)z_i}{\xi(1-\alpha) + (1-\xi)(1-\alpha)\frac{K_{i,1}}{K_{i,2}} + K_{i,1}\alpha} = 0$$

4.7.1 BUBBLE-POINT CALCULATIONS

At the bubble point, $V = 0$, $F = V + L_1 + L_2 = L_1 + L_2$, and $y_i = K_{i,1}x_{i,1} = K_{i,2}x_{i,2}$. The combination of the component mass balance and summing relations yields

$$z_i = x_{i,1} \left[\xi + (1-\xi)\frac{K_{i,1}}{K_{i,2}} \right], \quad \sum_{i=1}^{n_c} x_{i,1} - \sum_{i=1}^{n_c} y_i = 0, \quad \sum_{i=1}^{n_c} x_{i,1} - \sum_{i=1}^{n_c} x_{i,2} = 0$$

4.7.2 DEW-POINT CALCULATIONS

At the dew point, $L_1 = L_2 = 0$, $V = F$, $y_i = z_i$, and we have

$$y_i = K_i x_i, \quad \sum_{i=1}^{n_c} x_i = 1$$

where $x_i = x_{i,1}$.

4.7.3 ISOTHERMAL FLASH CALCULATIONS IN THE TWO-PHASE REGION

Introduction of $L = F - V$ and $\alpha = V/F$ into the component mass balance gives

$$x_i = \frac{z_i}{(1-\alpha) + \alpha K_i}$$

Example 4.20: Water–Isobutanol Equilibrium Calculations⁴³

A mixture of isobutanol(1) (20 mol%) and water(2) (80 mol%) is heated to the bubble point at a constant pressure of 1 atm. We assume that Raoult's law, $K_{i,j} = \gamma_{i,j} P_i / P$, can be applied. The vapor pressure of species i , P_i , for each component is given by the Antoine equation:

$$\log(P_1) = 7.62231 - \frac{1417.9}{191.15 + T} \quad \text{and} \quad \log(P_2) = 8.10765 - \frac{1750.29}{235 + T} \quad (T: ^\circ\text{C})$$

Activity coefficients of isobutanol(1) and water(2) are given by

$$\log \gamma_{1,j} = \frac{1.7x_{2,j}^2}{(2.43x_{1,j} + x_{2,j})^2} \quad \text{and} \quad \log \gamma_{2,j} = \frac{0.7x_{1,j}^2}{(x_{1,j} + 0.412x_{2,j})^2} \quad (j: \text{phase})$$

Determine the bubble-point and the dew-point temperatures at 1 atm. Plot the fraction evaporated (α) as a function of the boiling temperature between these two temperatures.

Solution

Bubble-point temperature. From the relation $z_i = x_{i,1} \left[\xi + (1-\xi) \frac{K_{i,1}}{K_{i,2}} \right]$, two nonlinear equations are obtained: $f(x_{i,1}) = x_{i,1} - \left[\frac{z_i}{\xi + (1-\xi) \frac{K_{i,1}}{K_{i,2}}} \right] = 0 (i = 1, 2)$.

From $K_{i,1}x_{i,1} = K_{i,2}x_{i,2}$, another two equations are obtained: $f(x_{i,2}) = x_{i,2} - x_{i,1} \frac{K_{i,1}}{K_{i,2}} = 0 (i = 1, 2)$.

Sum of mole fractions gives $f(\xi) = \sum_{i=1}^2 (x_{i,1} - x_{i,2}) = 0$.

Using $y_i = K_{i,1}x_{i,1}$ and $\sum_{i=1}^{n_c} x_{i,1} - \sum_{i=1}^{n_c} y_i = 0$, we have $f(T) = \sum_{i=1}^2 x_{i,1} (1 - K_{i,1}) = 0$.

Additional equations required are $P_1 = 10^{7.62231 - \frac{1417.9}{191.15 + T}}$, $P_2 = 10^{8.10765 - \frac{1750.29}{235 + T}}$, $\gamma_{1,j} = 10^{\frac{1.7x_{2,j}^2}{(2.43x_{1,j} + x_{2,j})^2}}$,

$$\gamma_{2,j} = 10^{\frac{0.7x_{1,j}^2}{(x_{1,j} + 0.412x_{2,j})^2}}, K_{i,j} = \frac{\gamma_{i,j} P_i}{P}$$

Initial guesses for solutions are set to

$$x_{1,1}(0) = x_{2,2}(0) = 0, \quad x_{2,1}(0) = x_{1,2}(0) = 1, \quad T(0) = 100, \quad \beta(0) = 0.8$$

The MATLAB function *ibweqf* defines the set of nonlinear equations.

```
function f = ibweqf(t, z, P)
% t(1)=x11, t(2)=x21, t(3)=x12, t(4)=x22, t(5)=T, t(6)=beta
% Required relations
P1 = 10^(7.62231 - 1417.9/(191.15 + t(5)));
P2 = 10^(8.10765 - 1750.29/(235 + t(5)));
gam11 = 10^(1.7*t(2)^2/((2.43*t(1) + t(2))^2));
gam12 = 10^(1.7*t(4)^2/((2.43*t(3) + t(4))^2));
gam21 = 10^(0.7*t(1)^2/((t(1) + 0.412*t(2))^2));
```

```

gam22 = 10^(0.7*t(3)^2/((t(3) + 0.412*t(4))^2));
k11 = gam11*P1/P; k12 = gam12*P1/P;
k21 = gam21*P2/P; k22 = gam22*P2/P;
% Nonlinear equations
f(1,1) = t(1) - z(1)/(t(6) + (1-t(6))*k11/k12); % i=1
f(2,1) = t(2) - z(2)/(t(6) + (1-t(6))*k21/k22); % i=2
f(3,1) = t(3) - t(1)*k11/k12;
f(4,1) = t(4) - t(2)*k21/k22;
f(5,1) = t(1) + t(2) - t(3) - t(4);
f(6,1) = t(1)*(1-k11) + t(2)*(1-k21);
end

```

The following commands set initial conditions and initial guesses for solutions and call the function *ibweqf* to solve the nonlinear equations using the built-in function *fsolve*.

```

>> x110 = 0; x220 = 0; x210 = 1; x120 = 1; T0 = 100; beta0 = 0.8;
>> z = [0.2 0.8]; P = 760; t0 = [x110 x210 x120 x220 T0 beta0];
>> [t,fval] = fsolve(@ibweqf, t0, [], z, P)
t =
0.0227 0.9773 0.6866 0.3134 88.5396 0.7329
fval =
1.0e-09 *
-0.0071
0.0392
0.5486
-0.0574
0
0.1771

```

We can see that

$$x_{11} = 0.0227, \quad x_{21} = 0.9773, \quad x_{12} = 0.6866, \quad x_{22} = 0.3134, \quad T = 88.5396^\circ\text{C}, \quad \beta = 0.7329$$

The vector *fval* in the results, whose elements converge to zeros, depicts function values of nonlinear equations.

Dew-point temperature. From $y_i = K_i x_i$, we have $f(x_i) = x_i - \frac{y_i}{K_i} = 0$ ($i = 1, 2$).

From $\sum_{i=1}^2 x_i = 1$, we have $f(T) = x_1 + x_2 - 1 = 0$.

Accompanying relations: $P_1 = 10^{\frac{7.62231 - 1417.9}{191.15 + T}}$, $P_2 = 10^{\frac{8.10765 - 1750.29}{235 + T}}$, $\gamma_1 = 10^{\frac{1.7 x_2^2}{(2.43 x_1 + x_2)^2}}$, $\gamma_2 = 10^{\frac{0.7 x_1^2}{(x_1 + 0.412 x_2)^2}}$, $K_i = \frac{\gamma_i P_i}{P}$ ($i = 1, 2$).

Initial guesses are set to $x_1(0) = 0.2$, $x_2(0) = 0.8$, $T(0) = 100$. The set of nonlinear equations is defined by the function *ibweqf2*.

```

function f = ibweqf2(t, y, P)
% t(1)=x1, t(2)=x2, t(3)=T
% Required relations
P1 = 10^(7.62231 - 1417.9/(191.15 + t(3)));
P2 = 10^(8.10765 - 1750.29/(235 + t(3)));
gam1 = 10^(1.7*t(2)^2/((2.43*t(1) + t(2))^2));
gam2 = 10^(0.7*t(1)^2/((t(1) + 0.412*t(2))^2));
k1 = gam1*P1/P; k2 = gam2*P2/P;
% Nonlinear equations
f = [t(1) - y(1)/k1;
      t(2) - y(2)/k2;
      t(1) + t(2) - 1];
end

```

The following commands set initial conditions and initial guesses for solutions and call the function *ibweqf2* to solve the nonlinear equations using the built-in function *fsolve*.

```

>> x10 = 0.2; x20 = 0.8; T0 = 100; y = [0.2 0.8]; P = 760; t0 = [x10 x20 T0];
>> t = fsolve(@ibweqf2, t0, [], y, P)
t =
0.0079 0.9921 93.9671

```

We can see that $x_1 = 0.0079$, $x_2 = 0.9921$, and $T = 93.9671^\circ\text{C}$.

PROBLEMS

- For ethane, the critical temperature and pressure are 305.3 K and 48.08 atm, respectively, and the acentric factor is 0.1. Use the virial equation of state to plot the compressibility factor Z versus the pressure in the range of $24.04 \leq P \leq 480.8$ (i.e., $0.5 \leq P_r \leq 10$, P : atm) for $T = 457.95$, 610.6 , and 3053 K (i.e., $T_r = 1.5, 2, 10$).
- Estimate the compressibility factor Z for isopropanol vapor at 200°C and 10 bar by the Lee–Kesler equation.⁴⁴ The critical temperature and pressure of isopropanol are 508.3 K and 47.62 bar, respectively, and the acentric factor of isopropanol is 0.668.
- Calculate the compressibility factor Z for carbon dioxide in the range of $0.5 \leq P_r \leq 1$ for $T_r = 1, 1.4, 1.8$, and 2 by the van der Waals equation of state. Plot Z versus P_r for each value of $T_r = 1, 1.4, 1.8$, and 2 . For carbon dioxide, the critical temperature and pressure are 304.2 K and 72.9 atm, respectively.
- Use the Peng–Robinson equation of state to calculate the specific volume (cm^3/g) for CO_2 at 310 K and at $P = 8$ and 75 bar. Compare the results with the experimental values of 70.58 and 3.90, respectively. For CO_2 , the molecular weight is 44 and $w = 0.228$, $T_c = 304.2$ K, $P_c = 73.82$ bar.⁴⁵
- Estimate the enthalpy departure $H^R = H - H^{ig}$ and entropy departure $S^R = S - S^{ig}$ for 1-butene vapor at 70 bar and 473.15 K. Use the virial state of equation. For 1-butene, $T_c = 420$ K, $P_c = 40.43$ bar, and $w = 0.191$.⁴⁶
- Table P4.6** shows experimental values for the enthalpy departure of isobutane at 175°C .⁴⁷ Calculate the theoretical values of enthalpy departure by the Peng–Robinson equation and compare the results with the experimental values shown in **Table P4.6**. For isobutane, the critical temperature and pressure are 408.1 K and 36.48 bar, respectively, and the acentric factor is 0.181.
- Calculate the enthalpy for methane(1)/ethane(2)/propane(3) system at -158 K and 6.8947 bar by cubic equations of state (Redlich–Kwong, Soave–Redlich–Kwong, and Peng–Robinson). Vapor-phase mole fractions and properties for each component are shown in **Table P4.7(1)** where ΔH_f and ΔG_f are in J/mol .

Table P4.7(2) shows constants for the heat capacity equation

$$C_p = a_1 + a_2 T + a_3 T^2 + a_4 T^3 + a_5 T^4 \quad (\text{Btu / lb mol/}^\circ\text{F}, T: {}^\circ\text{F})$$

TABLE P4.6
**Experimental Values for the Enthalpy
Departure of Isobutane**

P (atm)	10	20	35	70
$H - H_{ig}$ (J/g)	-15.4	-32.8	-64.72	-177.5

Source: Elliott, J.R. and Lira, C.T., *Introductory Chemical Engineering Thermodynamics*, 2nd ed., Prentice-Hall, Boston, MA, 2012, p. 329.

TABLE P4.7(1)
Properties for Each Component

Component	x (Mole Fraction)	w	T_c (K)	P_c (bar)	ΔH_f	ΔG_f
Methane	0.9852	0.012	190.6	45.99	-74,520	-50,460
Ethane	0.01449	0.100	305.3	48.72	-83,820	-31,855
Propane	0.000312	0.152	369.8	42.48	-104,680	-24,290

Source: Smith, J.M. et al., *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, 2005, pp. 680–686.

TABLE P4.7(2)
Constants for the Heat Capacity Equation

Component	a_1	$a_2 \times 10^2$	$a_3 \times 10^5$	$a_4 \times 10^8$	$a_5 \times 10^{11}$
Methane	8.245223	0.380633	0.8864745	-0.746115	0.182296
Ethane	11.51606	1.40309	0.854034	-1.106078	0.31622
Propane	15.58683	2.504953	1.404258	-3.52626	1.864467

Source: Henley, E.J. and Seader, J.D., *Equilibrium-Stage Separation Operations in Chemical Engineering*, John Wiley & Sons, Hoboken, NJ, 1981, pp. 720–724.

- 4.8 Table P4.8 shows experimental measurements on compressibility factors at various pressures for ammonia at 100°C. Determine fugacity coefficients from the data given in Table P4.8 at each pressure using a numerical integration scheme. In this case, we can use the MATLAB built-in function *cumtrapz*. Calculate the fugacity coefficients at each pressure from cubic equations of state, and compare the results with those obtained from the numerical integration by plotting both results versus pressure.
- 4.9 The temperature of the azeotrope for ethanol(1)/benzene(2) system at $P = 760$ mmHg is 68.2°C, and the composition at the azeotrope is 32.4% ethanol and 67.6% benzene (% by weight).

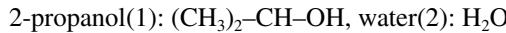
TABLE P4.8
Experimental Z Data for Ammonia

P (atm)	Z	P (atm)	Z	P (atm)	Z
1.374	0.0028	30.47	0.8471	300	0.3212
3.537	0.9828	33.21	0.831	400	0.4145
5.832	0.9728	36.47	0.8111	500	0.506
8.632	0.9599	40.41	0.7864	600	0.5955
11.352	0.9468	45.19	0.7538	700	0.6828
14.567	0.9315	51.09	0.7102	800	0.7684
19.109	0.9085	58.28	0.6481	900	0.8507
22.84	0.889	100	0.1158	1000	0.9333
26.12	0.8714	200	0.2221	1100	1.014

Source: Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 265.

The vapor pressures of ethanol and benzene may be obtained by the Antoine equation $\log P_i = A_i - \frac{B_i}{T + C_i}$ where T is the azeotrope temperature ($^{\circ}\text{C}$) and P_i is the vapor pressure (mmHg). For ethanol, $A_1 = 8.04494$, $B_1 = 1554.3$, and $C_1 = 222.65$, and for benzene, $A_2 = 6.90565$, $B_2 = 1211.033$, and $C_2 = 220.79$. The molecular weights of ethanol and benzene are 46.07 and 78, respectively. Determine the Wilson equation coefficients for this system.

- 4.10** Estimate the activity coefficients γ_1 and γ_2 for 2-propanol(1)/water(2) system by UNIFAC equation at $T = 353.52$ K. The liquid-phase compositions are $x_1 = 0.6854$ and $x_2 = 0.3146$. Functional groups for each component can be represented as



- 4.11** A mixture of isobutanol(1) (20 mol%) and water(2) (80 mol%) is heated to the bubble point at $P = 1$ atm. The bubble temperature for this system at $P = 1$ atm is known to be $T_b = 88.5396^{\circ}\text{C}$. Assume that this system follows Raoult's law, $k_i = \gamma_i P_i/P$ ($i = 1, 2$). The vapor pressure of species i , P_i , for each component is given by the Antoine equation:

$$\log(P_1) = 7.62231 - \frac{1417.9}{191.15 + T} \quad \text{and} \quad \log(P_2) = 8.10765 - \frac{1750.29}{235 + T} \quad (T: ^{\circ}\text{C})$$

Activity coefficients of isobutanol(1) and water(2) are given by

$$\log(\gamma_{1,j}) = \frac{1.7x_{2,j}^2}{(2.43x_{1,j} + x_{2,j})^2} \quad \text{and} \quad \log \gamma_{2,j} = \frac{0.7x_{1,j}^2}{(x_{1,j} + 0.412x_{2,j})^2} \quad (j: \text{phase})$$

Determine the fraction evaporated ($\alpha = V/F$) and mole fraction of each component at the boiling temperature.

- 4.12** Perform flash calculations for a 5-component system consisting of methane/ethane/propane/*n*-butane/*n*-pentane. The feed stream is fed into the flash drum at 40°C . The vapor pressure can be estimated using the Antoine equation $\log P_i^{\text{sat}} = A_i - \frac{B_i}{T + C_i}$ (P : mmHg, T : $^{\circ}\text{C}$). Feed compositions and parameters for the Antoine equation are given in Table P4.12. Determine the dew-point and bubble-point temperatures of the feed stream. Calculate the mole fractions of effluent vapor and liquid streams and the fraction evaporated ($\alpha = V/F$) for $P = 18, 22$, and 26 atm.
- 4.13** The feed consisting of acetone(1)/acetonitrile(2)/nitromethane(3) enters a flash drum operating at 80°C and 110 kPa. The feed composition is $z_1 = 0.45$, $z_2 = 0.35$, $z_3 = 0.20$. Assuming that Raoult's law is appropriate to this system, calculate V , L , x_i , and y_i . The vapor pressures (kPa) of the pure species at 80°C are $P_1^{\text{sat}} = 195.75$, $P_2^{\text{sat}} = 97.84$, $P_3^{\text{sat}} = 50.32$.⁴⁹

TABLE P4.12
Constants for the Antoine Equation

Component	Feed Composition	Constants		
		A	B	C
Methane	0.08	6.64380	395.74	266.681
Ethane	0.21	6.82915	663.72	256.681
Propane	0.38	6.80338	804.00	247.040
<i>n</i> -Butane	0.20	6.80776	935.77	238.789
<i>n</i> -Pentane	0.13	6.85296	1064.84	232.012

REFERENCES

1. Lim, K. H., *Thermodynamics for Engineers and Scientists*, Jayoo Academi, Busan, Korea, p. 338, 2011.
2. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, p. 102, 2005.
3. Lim, K. H., *Thermodynamics for Engineers and Scientists*, Jayoo Academi, Busan, Korea, p. 340, 2011.
4. Lee, B. I. and M. G. Kesler, *AIChE Journal*, 21(3), 510, 1975.
5. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, p. 23, 1985.
6. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, p. 98, 2005.
7. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, pp. 98–99, 2005.
8. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, p. 99, 2005.
9. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, pp. 131–132, 2005.
10. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, pp. 219–220, 2005.
11. Lim, K. H., *Thermodynamics for Engineers and Scientists*, Jayoo Academi, Busan, Korea, pp. 389–393, 2011.
12. Elliott, J. R. and C. T. Lira, *Introductory Chemical Engineering Thermodynamics*, 2nd ed., Prentice-Hall, Boston, MA, pp. 314–315, 2012.
13. Elliott, J. R. and C. T. Lira, *Introductory Chemical Engineering Thermodynamics*, 2nd ed., Prentice-Hall, Boston, MA, p. 314, 2012.
14. Henley, E. J. and J. D. Seader, *Equilibrium-Stage Separation Operations in Chemical Engineering*, John Wiley & Sons, Hoboken, NJ, pp. 720–724, 1981.
15. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, pp. 680–686, 2005.
16. Elliott, J. R. and C. T. Lira, *Introductory Chemical Engineering Thermodynamics*, 2nd ed., Prentice-Hall, Boston, MA, p. 352, 2012.
17. Elliott, J. R. and C. T. Lira, *Introductory Chemical Engineering Thermodynamics*, 2nd ed., Prentice-Hall, Boston, MA, p. 591, 2012.
18. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, p. 563, 2005.
19. Elliott, J. R. and C. T. Lira, *Introductory Chemical Engineering Thermodynamics*, 2nd ed., Prentice-Hall, Boston, MA, pp. 587–588, 2012.
20. Poling, B. E., J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, pp. 7.9–7.10, 2001.
21. Sandler, S. I., *Chemical, Biochemical and Engineering Thermodynamics*, 4th ed., John Wiley & Sons, Inc., Hoboken, NJ, p. 320, 2006.
22. Sandler, S. I., *Chemical, Biochemical and Engineering Thermodynamics*, 4th ed., John Wiley & Sons, Inc., Hoboken, NJ, p. 321, 2006.
23. Sandler, S. I., *Chemical, Biochemical and Engineering Thermodynamics*, 4th ed., John Wiley & Sons, Inc., Hoboken, NJ, p. 319, 2006.
24. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 274, 2008.
25. *Internal Critical Tables*, 1st ed., Vol. III, McGraw-Hill, New York, NY, p. 287, 1928.
26. Sandler, S. I., *Chemical, Biochemical and Engineering Thermodynamics*, 4th ed., John Wiley & Sons, Inc., Hoboken, NJ, p. 521, 2006.
27. Sandler, S. I., *Chemical, Biochemical and Engineering Thermodynamics*, 4th ed., John Wiley & Sons, Inc., Hoboken, NJ, p. 433, 2006.
28. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, pp. 791–792, 2005.
29. Poling, B. E., J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, pp. 8.78–8.81, 2001.
30. Lim, K. H., *Thermodynamics for Engineers and Scientists*, Jayoo Academi, Busan, Korea, pp. 832–833, 2011.

31. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, pp. 793–795, 2005.
32. Poling, B. E., J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*, 5th ed., McGraw-Hill, New York, NY, pp. 8.82–8.93, 2001.
33. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, pp. 795–797, 2005.
34. Elliott, J. R. and C. T. Lira, *Introductory Chemical Engineering Thermodynamics*, 2nd ed., Prentice-Hall, Boston, MA, p. 376, 2012.
35. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, p. 352, 2005.
36. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., pp. 359–363, McGraw-Hill, New York, NY, 2005.
37. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 268, 2008.
38. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, pp. 547–550, 2005.
39. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, p. 550, 2005.
40. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, p. 551, 2005.
41. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, p. 554, 2005.
42. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 528, 2008.
43. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 527–534, 2008.
44. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, pp. 89–90, 2005.
45. Elliott, J. R. and C. T. Lira, *Introductory Chemical Engineering Thermodynamics*, 2nd ed., Prentice-Hall, Boston, MA, p. 268, 2012.
46. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, p. 235, 2005.
47. Elliott, J. R. and C. T. Lira, *Introductory Chemical Engineering Thermodynamics*, 2nd ed., Prentice-Hall, Boston, MA, p. 329, 2012.
48. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 265, 2008.
49. Smith, J. M., H. C. Van Ness, and M. M. Abbott, *Introduction to Chemical Engineering Thermodynamics*, 7th ed., McGraw-Hill, New York, NY, pp. 368–369, 2005.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

5 Fluid Mechanics

Transportation of fluids forms an integral part of chemical engineering operations. The flow situations vary from simple pipe flow of a Newtonian fluid to non-Newtonian flow systems, and further, to a network of pipelines. In fluid-flow problems, mathematical calculations are used to determine the size of pipe, the fluid transport properties, the flow characteristics, and the energy that must be applied to move the fluid. But, because of the complexity of the flow pattern, most flows are described by a set of empirical or semiempirical equations, and only a few special problems can be entirely solved by rational, mathematical means. Many empirical equations have been proposed for solving problems of fluid flow in pipes. These equations relate the pressure loss in the flow system as a function of flow rate, pipe geometry, and the fluid's physical properties. And most of the equations describing fluid mechanics problems are nonlinear and therefore require trial and error computations, which are problems especially well suited for MATLAB®.

This chapter briefly covers calculation methods for basic flow problems, pressure losses in pipes and fittings for a variety of fluid flows that include single- and two-phase flows, compressible flow, and non-Newtonian fluid flows. The approach taken here emphasizes the engineering problem-solving, with little reference to the mechanisms of fluid mechanics. The main objective of this chapter is to provide readers with various fluid-flow models and corresponding MATLAB programs that have already been found, or can potentially be used, for academic or industrial applications. The MATLAB programs can be used in undergraduate or graduate courses on fluid mechanics. Researchers and practicing engineers in the field of process engineering can apply these MATLAB programs in modeling and simulation of chemical processes. Throughout this chapter, emphasis is placed on the connection between physical reality and the mathematical models or correlations of reality.

5.1 LAMINAR FLOW

5.1.1 REYNOLDS NUMBER

The Reynolds number is defined by

$$N_{Re} = \frac{Dv\rho}{\mu}$$

where

D is the inside pipe diameter

v is the velocity

ρ is the density of the fluid

μ is the viscosity of the fluid

If the volumetric flow rate is given, N_{Re} can be represented as

$$N_{Re} = \frac{\rho v D}{\mu} = 50.6 \frac{Q\rho}{ud} = 6.31 \frac{W}{ud}$$

where

D (ft) and d (in.) are the inside pipe diameters

μ (cp) is the viscosity of the fluid

Q (gpm) and W (lb/hr) are the volumetric flow rate and mass flow rate, respectively

5.1.2 FLOW IN A HORIZONTAL PIPE

Consider an incompressible Newtonian fluid flowing at steady state inside a horizontal circular pipe at constant temperature. Figure 5.1 shows a schematic of this laminar flow in the x direction where the inside pipe radius is R .

The shear stress at the radius r , τ_{rx} , and the velocity, v_x , are given by¹

$$\tau_{rx} = \left(\frac{\Delta P}{2L} \right) r, \quad v_x = \frac{R^2 \Delta P}{4\mu L} \left[1 - \left(\frac{r}{R} \right)^2 \right]$$

where

$$\Delta P = P_0 - P_L$$

R is the inside pipe radius

L is the length of the pipe

The average velocity $v_{x,avg}$ is obtained from

$$v_{x,avg} = \frac{1}{\pi R^2} \int_0^R 2\pi r v_x dr = \frac{R^2 \Delta P}{8\mu L}$$

The MATLAB function *hzpipe* calculates the average velocity of the laminar flow inside a horizontal circular pipe and plots the velocity distribution inside the pipe. The basic syntax is

```
avgv = hzpipe(L, R, mu, delP)
```

where L (m) and R (m) are the length and the radius of the pipe, respectively, μ (kg/m/sec) is the viscosity of the fluid, ΔP (Pa) is the pressure drop, and $avgv$ (m/sec) is the average velocity.

```
function avgv = hzpipe(L, R, mu, delP)
% Average velocity and velocity distribution of the laminar flow
% inside a horizontal circular pipe
% inputs
% L: length of pipe (m)
% R: radius of pipe (m)
% mu: viscosity (kg/m/s)
% delP: pressure drop (Pa)
% output
% avgv: average velocity (m/s)
r = linspace(-R, R, 100);
v = delP*R.^2.* (1 - (r/R).^2) / (4*mu*L);
avgv = delP*R.^2 / (8*mu*L);
plot(v, r), grid, axis([0 1.5 -R, R])
xlabel('v_x(r)'), ylabel('r'), axis([0 1 -R R])
end
```

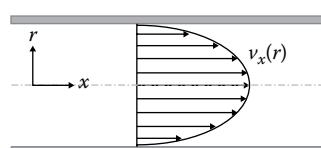


FIGURE 5.1 Laminar flow of a Newtonian fluid in a horizontal pipe.

Example 5.1: Newtonian Fluid Flow Inside a Horizontal Pipe

Water flows inside a horizontal circular pipe at 25°C. Determine the average velocity and plot the velocity distribution inside the pipe. The length and the inside radius of the pipe are 15 and 0.009 m, respectively, the viscosity of water is 8.937×10^{-4} kg/(m·sec), and the pressure drop is $\Delta P = 520$ Pa.

Solution

The following commands produce desired outputs and illustrate velocity distribution as shown in Figure 5.2:

```
>> L = 15; R = 0.009; mu = 8.937e-4; delP = 520;
>> Vavg = hzpipe(L,R,mu,delP)
Vavg =
0.3927
>> axis([0 1 -R R])
```

5.1.3 LAMINAR FLOW IN A HORIZONTAL ANNULUS

An incompressible fluid is flowing within an annulus between two concentric horizontal pipes as shown in Figure 5.3.² For a Newtonian fluid, the velocity distribution v_x and the average velocity v_{avg} can be expressed as³

$$v_x = \frac{\Delta P}{4\mu L} \left[R_2^2 - r^2 + \frac{R_1^2}{\ln(R_2/R_1)} \left(\ln \frac{r}{R_2} \right) \right]$$

$$v_{avg} = \frac{\Delta P}{8\mu L} \left[R_1^2 + R_2^2 - \frac{R_2^2 - R_1^2}{\ln(R_2/R_1)} \right]$$

where

$$\Delta P = P_0 - P_L$$

R_1 and R_2 are the inside radius of inner and outer pipes, respectively

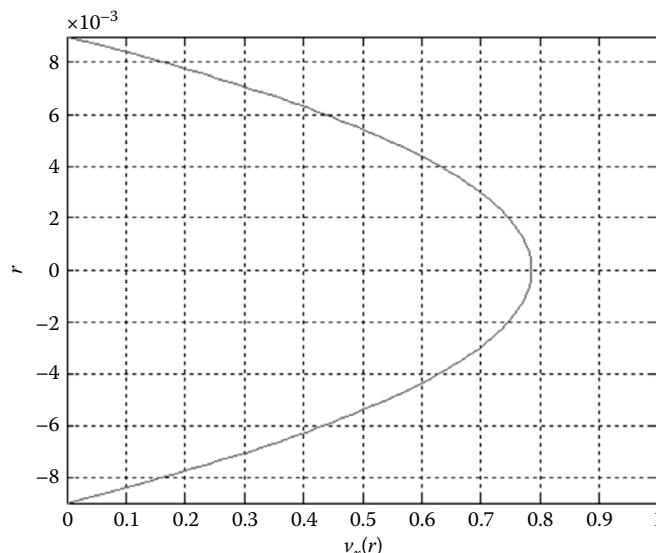


FIGURE 5.2 Velocity distribution inside a circular pipe.

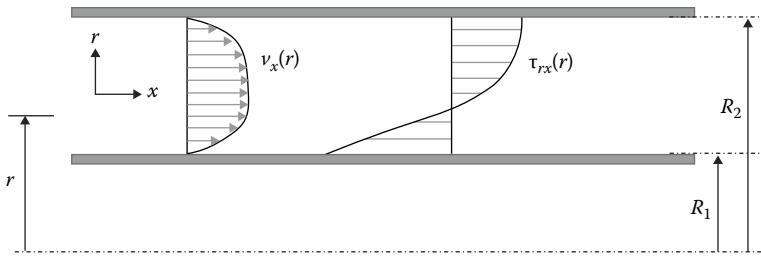


FIGURE 5.3 Laminar flow in a horizontal annulus. (Modified from Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 294.)

For a Newtonian fluid, the shear stress τ_{rx} is given by

$$\tau_{rx} = -\mu \frac{dv_x}{dr}$$

For some non-Newtonian fluids, the shear stress τ_{rx} can be represented as

$$\tau_{rx} = -K \left| \frac{dv_x}{dr} \right|^{n-1} \frac{dv_x}{dr}$$

For a Newtonian laminar flow in a horizontal annulus, the velocity distribution and the average velocity are calculated by the MATLAB function *hzannpipe*. The syntax is

```
avgv = hzannpipe(L,R1,R2,mu,delP)
```

This function generates plots showing distributions of the velocity and the shear stress.

```
function avgv = hzannpipe(L,R1,R2,mu,delP)
% Calculates the average velocity and plots velocity and shear stress
% distribution for Newtonian laminar flow in a horizontal annulus.
% input
% L: length of pipe (m)
% R1,R2: inside radius of inner and outer pipe (m)
% mu: viscosity (kg/m/s)
% delP: pressure drop (Pa)
% output
% avgv: average velocity (m/s)

n = 100; r = linspace(R1,R2,n); h = (R2-R1)/n;
v = delP*(R2^2 - r.^2 + (R2^2 - R1^2)/(log(R2/R1)).*log(r/R2))./(4*mu*L);
avgv = delP*(R1.^2 + R2.^2 - (R2.^2 - R1.^2)./(log(R2./R1)))./(8*mu*L);
dv = diff(v); dv = [dv dv(end)]; taurx = -mu.*dv./h;
subplot(1,2,1)
plot(v,r), grid, axis([0 0.2 R1 R2]), xlabel('v_x(r)'), ylabel('r')
subplot(1,2,2)
plot(taurx,r), grid, axis tight
xlabel('\tau_{rx}(r)'), ylabel('r')
end
```

Example 5.2: Laminar Flow of Water in a Horizontal Annulus

Water flows inside a horizontal annulus at 25°C. Determine the average velocity and plot the velocity and the shear stress distributions inside the annulus. The length and the inside radius of the inner and outer pipes are 12, 0.025, and 0.036 m, respectively. The viscosity of water is 8.937×10^{-4} kg/(m · sec), and the pressure drop is $\Delta P = 110$ Pa.

Solution

The following commands call the function *hzannpipe* to produce plots of the velocity and the shear stress distributions as shown in Figure 5.4:

```
>> mu = 8.937e-4; delP = 110; L = 12; R1 = 0.025; R2 = 0.036;
>> avgv = hzannpipe(L,R1,R2,mu,delP)
avgv =
0.1037
```

5.1.4 VERTICAL LAMINAR FLOW OF A FALLING FILM

Consider a liquid flowing down a vertical surface. The liquid is a fully established film where the velocity profile in the film does not vary in the direction of flow as shown in Figure 5.5.

From the z -momentum balance, the differential equation for the momentum flux is given by⁴

$$\frac{d\tau_{xz}}{dz} = \rho g, \quad \tau_{xz}(x = 0) = 0$$

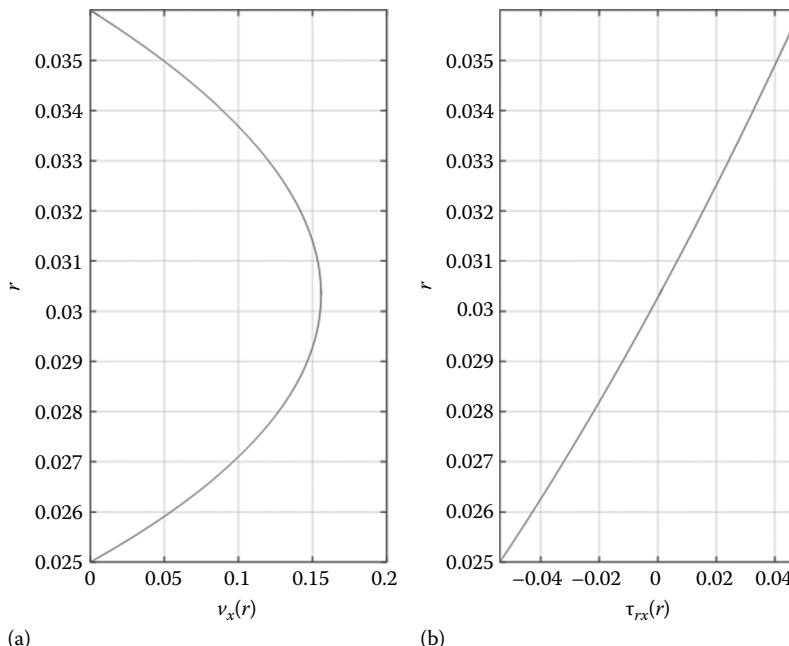


FIGURE 5.4 (a) Velocity ($v_x(r)$) and (b) shear stress ($\tau_{rx}(r)$) distributions for a laminar flow in a horizontal annulus.

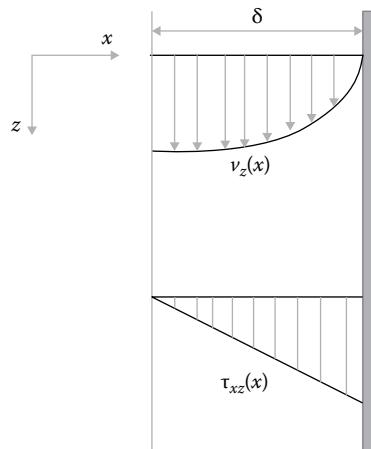


FIGURE 5.5 Vertical laminar flow of a liquid film.

For a Newtonian fluid, the momentum flux (shear stress) is related to the velocity gradient by

$$\tau_{xz} = -\mu \frac{dv_z}{dx} \quad (x = \delta \text{ to } v_z = 0)$$

Integration of the differential equation followed by substitution of boundary conditions yields the velocity distribution v_z and the average velocity $v_{z,avg}$ as

$$v_z = \frac{\rho g \delta^2}{2\mu} \left[1 - \left(\frac{x}{\delta} \right)^2 \right], \quad v_{z,avg} = \frac{\rho g \delta^2}{3\mu}$$

The MATLAB function *vtwall* calculates the velocity distribution and the average velocity for a Newtonian fluid. The syntax is

```
[avgv] = vtwall(rho, mu, delta)
```

where rho and mu are the density and the viscosity of the fluid, respectively, and delta is the film thickness.

```
function [avgv] = vtwall(rho, mu, delta)
% Calculates velocity distribution and average velocity for vertical
% laminar flow of a falling film
% input
% rho: density (kg/m^3), mu: viscosity (kg/m/s)
% delta: film thickness (m)
% output
% avgv: average velocity (m/s)
g = 9.8; h = delta/100; x = [0:h:delta]; Rg = rho*g*delta^2/2/mu;
v = Rg.* (1 - (x/delta).^2); avgv = Rg*2/3;
plot(x,v), grid, ylabel('v_z(x)'), xlabel('x'), axis([0 delta 0 max(v)])
end
```

Example 5.3: Velocity Distribution of a Falling Film

Plot the velocity distribution $v_z(x)$ and calculate the average velocity for a falling film of a Newtonian fluid with $\rho = 780 \text{ kg/m}^3$ and $\mu = 0.172 \text{ kg/(m}\cdot\text{sec)}$. The film thickness is $\delta = 0.0021 \text{ m}$.

Solution

The function `vtwall` produces the desired plot as shown in Figure 5.6.

```
>> rho = 780; mu = 0.172; delta = 0.0021;
>> [avgv] = vtwall(rho, mu, delta)
avgv =
0.0653
```

5.1.5 FALLING PARTICLES

The terminal velocity v_t of a falling spherical particle is given by

$$v_t = \sqrt{\frac{4g(\rho_p - \rho)D_p}{3C_D\rho}}$$

where

- v_t is the terminal velocity (m/sec)
- g is the acceleration of gravity ($=9.8 \text{ m/sec}^2$)
- ρ_p is the density of the particle (kg/m^3)
- ρ is the fluid density (kg/m^3)
- D_p is the diameter of the spherical particle (m)
- C_D is a dimensionless drag coefficient

Rearrangement of the terminal velocity equation gives

$$3C_D\rho v_t^2 - 4g(\rho_p - \rho)D_p = 0$$

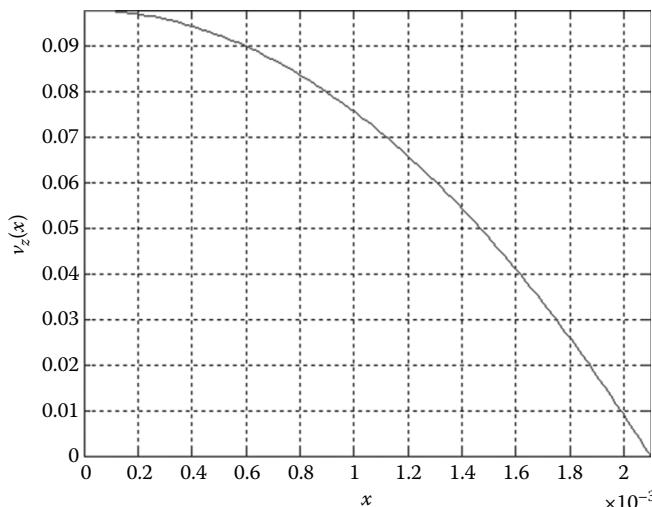


FIGURE 5.6 Velocity distribution for a falling film of a Newtonian fluid.

At the terminal velocity, the drag coefficient can be expressed as a function of the Reynolds number N_{Re} ⁵:

$$C_D = \begin{cases} \frac{24}{N_{Re}} & : N_{Re} < 0.1 \\ \frac{24}{N_{Re}} (1 + 0.14 N_{Re}^{0.7}) & : 0.1 \leq N_{Re} \leq 1,000 \\ 0.44 & : 1,000 < N_{Re} \leq 350,000 \\ 0.19 - \frac{8 \times 10^4}{N_{Re}} & : 350,000 < N_{Re} \end{cases}$$

where $N_{Re} = D_p v_t \rho / \mu$ and μ is the fluid viscosity (kg/(m · sec) or Pa · sec).

The nonlinear equation to be solved is defined by the MATLAB function *vtfun*. The syntax is

```
fvt = vtfun(x, w)
```

The required data are supplied by the structure *w*, the fields of which contain the densities of the particle and the fluid, the viscosity of the fluid, and the diameter of the particle.

```
function fvt = vtfun(x,w)
% x: unknown terminal velocity, w: data structure
g = 9.8; rp = w.rp; ro = w.ro; mu = w.mu; dp = w.dp;
Nre = dp*ro*x/mu; % Reynolds number
if Nre < 0.1
    Cd = 24./Nre;
elseif Nre < 1000
    Cd = (1 + 0.14*Nre.^0.7)*24./Nre;
elseif Nre < 3.5e5
    Cd = 0.44;
else
    Cd = 0.19 - 8e4./Nre;
end
fvt = 3*Cd*ro*x.^2 - 4*g*(rp - ro)*dp; % nonlinear equation
end
```

Example 5.4: Terminal Velocity of a Falling Particle

Estimate the terminal velocity for a spherical particle falling in water at 25°C. For the particle, $\rho_p = 1780 \text{ kg/m}^3$ and $D_p = 0.2 \text{ mm}$. The viscosity and the density of water are $\mu = 8.391 \times 10^{-4} \text{ kg/(m · sec)}$ and 994.6 kg/m^3 , respectively.

Solution

The nonlinear equation defined by the function *vtfun* can be solved by using the built-in function *fzero*. *vt0* is an initial guess for the terminal velocity. The following commands produce the terminal velocity (m/sec):

```
>> rp = 1780; ro = 994.6; dp = 2e-4; mu = 8.931e-4; vt0 = 1e-3;
>> w.rp = rp; w.ro = ro; w.mu = mu; w.dp = dp;
>> vt = fzero(@vtfun, vt0, [], w)
vt =
    0.0145
```

5.2 FRICTION FACTOR

The Fanning friction factor f is related to the Reynolds number N_{Re} through a set of correlations and depends on whether the flow regime is laminar, transitional, or turbulent. A simple relation between f and N_{Re} can be expressed as

$$f = aN_{Re}^b$$

where a and b are constants. The friction factor is affected by the roughness of the surface at a high Reynolds number ($N_{Re} \geq 2000$).

For laminar flow ($N_{Re} \leq 2000$), the Fanning friction factor f and the Darcy friction factor f_D are given by

$$f = \frac{16}{N_{Re}}, \quad f_D = \frac{64}{N_{Re}} = 4f$$

The Darcy friction factor is four times the Fanning friction factor (i.e., $f_D = 4f$).

For fully developed turbulent flow regime ($N_{Re} > 3000$) in rough pipes, the correlations can be expressed in terms of the surface roughness parameter ϵ/D . The Shacham equation⁶ (SC) is an explicit form given by

$$f = \frac{1}{16} \left[\log \left\{ \frac{\epsilon/D}{3.7} - \frac{5.02}{N_{Re}} \log \left(\frac{\epsilon/D}{3.7} + \frac{14.5}{N_{Re}} \right) \right\} \right]^{-2}$$

The equation given by Colebrook⁷ (CB) has received wide acceptance:

$$\frac{1}{\sqrt{f}} = -1.7372 \ln \left[\frac{\epsilon}{3.7D} + \frac{1.255}{N_{Re} \sqrt{f}} \right]$$

Another widely used implicit equation is the Colebrook–White equation⁸ (CBW) given by

$$\frac{1}{\sqrt{f}} = -4.0 \log \left(\frac{\epsilon}{D} + \frac{4.67}{N_{Re} \sqrt{f}} \right) + 2.28$$

The Haaland equation⁹ (HA) is an explicit equation given by

$$f = \left[-3.6 \log \left\{ \frac{6.9}{N_{Re}} + \left(\frac{\epsilon/D}{3.7} \right)^{\frac{10}{9}} \right\} \right]^{-2} \quad (4 \times 10^4 < N_{Re} < 10^8, 0 < \epsilon/D < 0.05)$$

The Chen equation¹⁰ (CH) is explicit and easier to use than the Colebrook equation. This can be expressed as

$$\frac{1}{\sqrt{f}} = -4 \log \left[\frac{\epsilon}{3.7D} - \frac{5.02}{N_{Re}} \log A \right], \quad A = \frac{\epsilon}{3.7D} + \left(\frac{6.7}{N_{Re}} \right)^{0.9}$$

TABLE 5.1
Values of Absolute Pipe Roughness

Pipe Material	ϵ (ft)
Riveted steel	0.003–0.03
Concrete	0.001–0.01
Wood stave	0.0006–0.003
Cast iron	0.00085
Galvanized iron	0.0005
Asphalted cast iron	0.0005
Commercial steel or wrought iron	0.00015
Drawn tubing	0.000005

Source: de Nevers, N., *Fluid Mechanics for Chemical Engineers*, 3rd ed., McGraw-Hill, New York, NY, 2005, p. 187.

For commercial steel pipes, the surface roughness is about $\epsilon = 0.00015$ ft. Table 5.1 shows the values of ϵ for various pipe materials.

As a single equation spanning all fluid-flow regimes and surface roughness parameter ϵ/D , the Churchill equation¹² is used:

$$f = 2 \left[\left(\frac{8}{N_{Re}} \right)^{12} + \frac{1}{(A+B)^{1.5}} \right]^{1/12}$$

$$A = (2.457 \ln C)^{16}, \quad B = \left(\frac{37,530}{N_{Re}} \right)^{16}, \quad C = \left(\frac{7}{N_{Re}} \right)^{0.9} + 0.27 \frac{\epsilon}{D}$$

For turbulent flow through smooth pipes where $\epsilon/D = 0$, the Nikuradse equation¹³ (NK)

$$\frac{1}{\sqrt{f}} = 4.0 \log(N_{Re} \sqrt{f}) - 0.4$$

or the Blasius equation (BS)

$$f = 0.0791 N_{Re}^{-1/4}$$

can be used.

The MATLAB function *frfactor* calculates the friction factor for the given values of ϵ/D and N_{Re} .

```
function frfactor(eD, Nre)
% Friction factor correlations
% inputs:
% eD: surface roughness parameter
% Nre: Reynolds number
f0 = 5e-3; % initial guess
% Shacham eqn.
```

```

fSC = 1./(log10(eD/3.7 - (5.02./Nre).*log10(eD/3.7+14.5/Nre))).^2 /16;
% Colebrook eqn.
funCB = @(f) (1/sqrt(f) + 1.7372*log(eD/3.7 + 1.255/Nre/sqrt(f)));
fCB = fzero(funCB, f0);
% Colebrook-White eqn.
funCBW = @(f) (1/sqrt(f) + 4*log10(eD + 4.67/Nre/sqrt(f)) - 2.28);
fCBW = fzero(funCBW, f0);
% Haaland eqn.
fHA = 1./(-3.6*log10(6.9/Nre + (eD/3.7)^(10/9))).^2;
% Chen eqn.
Av = eD/3.7 + (6.7/Nre)^0.9;
fCH = 1./(-4*log10(eD/3.7 - 5.02*log10(Av)/Nre)).^2;
% Nikuradse eqn.
funNK = @(f) (1./sqrt(f) - 4*log10(Nre*sqrt(f)) + 0.4);
fNK = fzero(funNK, f0);
% Blasius eqn.
fBS = 0.0791*Nre^(-1/4);
fprintf('Shacham eqn.: f = %g\n', fSC);
fprintf('Colebrook eqn.: f = %g\n', fCB);
fprintf('Colebrook-White eqn.: f = %g\n', fCBW);
fprintf('Haaland eqn.: f = %g\n', fHA);
fprintf('Chen eqn.: f = %g\n', fCH);
fprintf('Nikuradse eqn.: f = %g\n', fNK);
fprintf('Blasius eqn.: f = %g\n', fBS);
end

```

Example 5.5: Friction Factor

Determine the friction factors using various correlation equations (i.e., SC, CB, CBW, HA, CH, NK, and BS equation) at $N_{Re} = 2 \times 10^4$ and $N_{Re} = 3.2 \times 10^7$ for smooth pipes where $\epsilon/D = 0$.

Solution

$$N_{Re} = 2 \times 10^4:$$

```

>> eD = 0; Nre = 2e4;
>> frfactor(eD, Nre)
Shacham eqn.: f = 0.00648922
Colebrook eqn.: f = 0.00647063
Colebrook-White eqn.: f = 0.00647323
Haaland eqn.: f = 0.00643718
Chen eqn.: f = 0.00648215
Nikuradse eqn.: f = 0.00647573
Blasius eqn.: f = 0.00665149

```

$$N_{Re} = 3.2 \times 10^7:$$

```

>> eD = 0; Nre = 3.2e7;
>> frfactor(eD, Nre)
Shacham eqn.: f = 0.0017349
Colebrook eqn.: f = 0.00172196
Colebrook-White eqn.: f = 0.00172236
Haaland eqn.: f = 0.0017363
Chen eqn.: f = 0.00172145
Nikuradse eqn.: f = 0.00172273
Blasius eqn.: f = 0.00105169

```

TABLE 5.2
Relation between Reynolds Number (N_{Re})
and Friction Factor (f)

N_{Re}	f
8,500	0.008
20,000	0.0065
30,000	0.006
60,000	0.005
700,000	0.003
1,000,000	0.0028
10,000,000	0.002

Source: Kapuno, R.R.A., *Programming for Chemical Engineers*, Infinity Science Press, Hingham, MA, 2008, p. 105.

Example 5.6: Friction Factor Correlation

Determine an empirical correlation to represent the relationship between the Reynolds number and the Fanning friction factor for a smooth pipe using the data given in Table 5.2.¹⁴

Solution

Assume a simple relation $f = aN_{Re}^b$. Taking logarithm on both sides, we have a linear equation:

$$\log(f) = \log(a) + b \log(N_{Re})$$

Constants a and b can be determined from simple linear regression. The built-in function *polyfit* is used to perform the linear regression calculation.

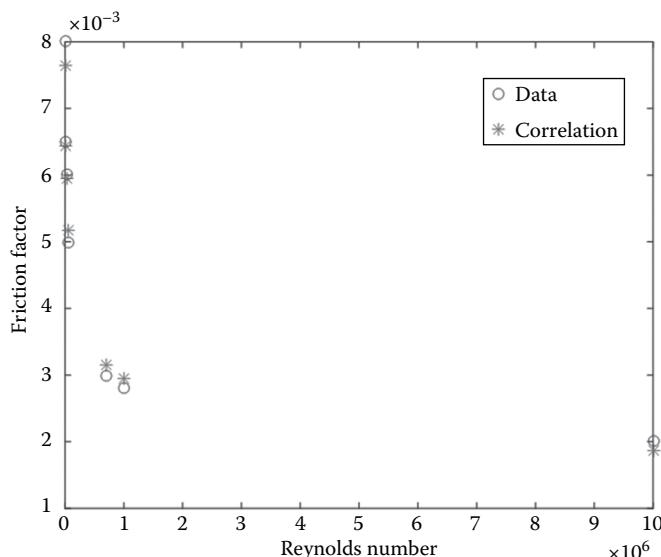


FIGURE 5.7 Reynolds number versus friction factor.

```

>> Nre = [8500 20000 30000 60000 700000 1000000 10000000];
>> f = [0.008 0.0065 0.006 0.005 0.003 0.0028 0.002];
>> x = log10(Nre); y = log10(f); p = polyfit(x,y,1);
>> a = 10^p(2); b = p(1); fv = a*Nre.^b;
>> plot(Nre, f, 'o', Nre, fv, '*'), legend('Data','Correlation')
>> xlabel('Reynolds number'), ylabel('Friction factor')
>> a, b
a =
0.0468
b =
-0.2001

```

The correlation is given by $f = aN_{Re}^b = 0.0468N_{Re}^{-0.2}$. The graph of this correlation and the data points are shown in [Figure 5.7](#).

5.3 FLOW OF FLUIDS IN PIPES

Application of the Bernoulli equation between two points in a pipeline where a fluid flows yields

$$\frac{1}{\rho} \int_1^2 dP + \frac{1}{g_c} \int_1^2 v dv + \int_1^2 dz = 0$$

Integration of this equation gives

$$\frac{1}{\rho} (P_2 - P_1) + \frac{v_2^2 - v_1^2}{2g_c} + (z_2 - z_1) = 0$$

or

$$\frac{g_c \Delta P}{\rho} + \frac{v_2^2 - v_1^2}{2} + g_c \Delta z = 0$$

where

P is the pressure of the fluid (lb_f/ft^2)

ρ is the density of the fluid (lb_m/ft^3)

v is the velocity of the fluid (ft/sec)

g_c is the dimensionless constant ($32.174 (\text{lb}_m/\text{lb}_f)(\text{ft/sec}^2)$)

z is the elevation of the fluid (ft)

the subscripts 1 and 2 denote the conditions at initial and final points, respectively

Incorporating the head loss due to friction, h_L , with constant pipe diameter (i.e., $v_1 = v_2$), the above relation becomes

$$\frac{\Delta P}{\rho} + \Delta z = h_L$$

5.3.1 FRICTION LOSS

The friction factor can be used to estimate the pressure drop, ΔP_f due to friction loss from

$$\Delta P_f = 2f \rho \frac{L v^2}{D g_c}$$

where

f is the dimensionless friction factor

L is the pipe length

The friction loss, F_f , for isothermal liquid flow in uniform circular pipes can be estimated by the relation

$$F_f = \frac{\Delta P_f}{\rho} = 2f \frac{Lv^2}{Dg_c} = f_D \frac{Lv^2}{2Dg_c}$$

The energy balance for incompressible flow in a pipe with diameter D (ft) and length L (ft) can be written as

$$-\frac{1}{2}v^2 + g\Delta z + \frac{g_c \Delta P}{\rho} + g_c F_f = -\frac{1}{2}v^2 + g\Delta z + \frac{g_c \Delta P}{\rho} + 2f \frac{Lv^2}{D} = 0$$

From this equation, the fluid velocity is given by

$$v = \sqrt{\frac{g\Delta z + g_c \Delta P / \rho}{1/2 - 2fL/D}}$$

If the fluid performs mechanical work, the energy balance gives

$$\frac{\Delta v^2}{2} + g\Delta z + \frac{\Delta P}{\rho} + F_f = -\frac{\dot{W}_s}{\dot{m}}$$

Example 5.7: Determination of Pipe Diameter

Water flows inside a smooth pipeline at 21°C with a flow rate of 0.0085 m³/sec. The length of the pipeline is 250 m, and the elevation of the upstream point is high enough to overcome the friction loss $F_f = 27.5$ J/kg. The Fanning friction factor is given by $= 0.0468N_{Re}^{-0.2}$, and the density and the viscosity of water at 21°C are $\rho = 997.91$ kg/m³ and $\mu = 0.000982$ kg/m·sec, respectively. Determine the diameter of the pipeline.

Solution

The flow rate equation $Q = Av = (\pi D^2/4)v$ can be solved for v to give $v = 4Q/\pi D^2$. Substitution of this velocity into the Fanning equation gives

$$f = 0.0468N_{Re}^{-0.2} = 0.0468 \left(\frac{Dv\rho}{\mu} \right)^{-0.2} = 0.0468 \left(\frac{4\rho Q}{\pi \mu D} \right)^{-0.2}$$

Substitution of this relation into the equation for the friction loss yields

$$F_f = \frac{\Delta P_f}{\rho} = 2f \frac{Lv^2}{Dg_c} = (2) \left(0.0468N_{Re}^{-0.2} \right) \frac{Lv^2}{Dg_c} = 0.0936 \left(\frac{4\rho Q}{\pi \mu D} \right)^{-0.2} \left(\frac{L}{Dg_c} \right) \left(\frac{4Q}{\pi D^2} \right)^2$$

Thus, the pipe diameter D (m) can be obtained from the solution of the nonlinear equation

$$g(D) = 0.0936 \left(\frac{4\rho Q}{\pi \mu D} \right)^{-0.2} \left(\frac{L}{Dg_c} \right) \left(\frac{4Q}{\pi D^2} \right)^2 - F_f = 0$$

The built-in function *fzero* can be used to solve the equation as follows:

```
>> rho = 997.92; Q = 0.0085; mu = 0.000982; L = 250; gc = 1; Ff = 27.5;
>> g = @(D) 0.0936*(4*rho*Q/(pi*mu*D))^{(-0.2)}*(L/D/gc)*(4*Q/pi/D^2)^2 - Ff;
>> D0 = 1;
>> D = fzero(g,D0)
D =
0.0995
```

Example 5.8: Flow in a Pipe¹⁵

Figure 5.8 shows a pipeline that delivers water at a constant temperature from point 1 to point 2. At point 1, the pressure and the elevation are $p_1 = 150$ psig and $z_1 = 0$ ft, respectively, and at point 2, the pressure p_2 is atmospheric and the elevation is $z_2 = 300$ ft. The density ρ (lb_m/ft^3) and viscosity μ ($\text{lb}_m/(\text{ft} \cdot \text{sec})$) of the water can be estimated from the following equations:

$$\rho = 62.122 + 0.0122T - 1.54 \times 10^{-4}T^2 + 2.65 \times 10^{-7}T^3 - 2.24 \times 10^{-10}T^4$$

$$\ln \mu = -11.0318 + \frac{1057.51}{T + 214.624}$$

where T is in °F.

1. Determine the volumetric flow rate (gal/min) at $T = 60^\circ\text{F}$ for a pipeline with an effective length of $L = 1000$ ft and made of nominal 8 in. schedule 40 commercial steel pipe ($\epsilon = 0.00015$ ft).
2. Estimate the flow velocities v in ft/sec and flow rates q in gal/min for pipelines at $T = 60^\circ\text{F}$ with effective lengths of $L = 500, 1,000, 1,500, \dots, 10,000$ ft and made of nominal 4, 5, 6, and 8 in. schedule 40 commercial steel pipe. Prepare plots of velocity v versus D and L , and flow rate q versus D and L .
3. Repeat part (1) at $T = 40^\circ\text{F}, 60^\circ\text{F}$, and 100°F and display the results in a table showing temperature, density, viscosity, and flow rates.

Solution

1. The general mechanical energy balance on an incompressible liquid applied to this problem yields

$$-\frac{1}{2}v^2 + g\Delta z + \frac{g_c \Delta P}{\rho} + 2f \frac{Lv^2}{D} = 0$$

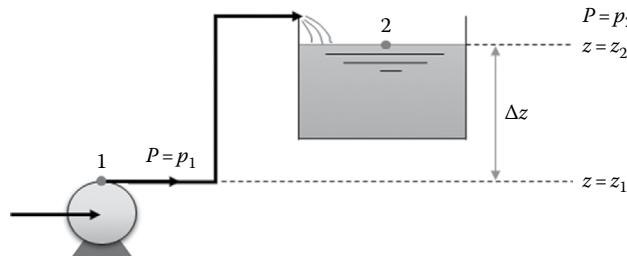


FIGURE 5.8 Water flow in a pipeline. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 110.)

The dimensionless friction factor f is a function of the Reynolds number N_{Re} , and N_{Re} is a function of the flow velocity v . The above equation can be solved for v to yield a nonlinear equation as

$$f(v) = v - \sqrt{\frac{g\Delta z + g_e \Delta P / \rho}{1/2 - 2fL/D}} = 0$$

This equation can be solved using the MATLAB built-in function *fzero*. Input data to be supplied to the function *fzero* consist of temperature T (°F), pipe length L (ft), inside diameter D (in.), pipe roughness rf (e t), the difference in elevation dz (ft), and the difference in pressure dP (psi). The inside diameter of the 8 in. schedule 40 steel pipe is 7.981 in., $= 0.00015$ ft, $dz = z_2 - z_1 = 300$ ft, and $dP = p_2 - p_1 = -150$ psi. The function *vwf* defines the nonlinear equation to be solved.

```
function fv = vwf(v,T,L,D,rf,dz,dP)
g = 32.174; gc = 32.174;
D = D/12; dP = 144*dP; % psi->lbf/ft^2
eD = rf./D; % roughness factor
% density(rho; lbm/ft^3) and viscosity(mu; lbm/ft/s) at T
rho = 62.122+0.0122*T-(1.54e-4)*T.^2+(2.65e-7)*T.^3-(2.24e-10)*T.^4;
mu = exp(-11.0318 + 1057.51./(T+214.624));
% velocity equation
Nre = D.*v.*rho./mu; % Reynolds number
if Nre < 2100
    f = 16./Nre;
else
    den = 16*(log10(eD/3.7 - 5.02*log10(eD/3.7+14.5./Nre))./Nre).^2;
    f = 1./den;
end
fv = v - sqrt((g*dz + gc*dP./rho)./(0.5 - 2*f.*L./D));
end
```

The following commands define data and call the function *fzero* to produce desired outputs:

```
>> clear all; T=60; L=1000; D=7.981; rf=0.00015; dz=300; dP=-150;
>> v0 = 10; v = fzero(@vwf, v0, [], T,L,D,rf,dz,dP)
>> q = (7.481*60)*(pi*v.* (D/12).^2)/4 % gpm
v =
    11.6133
q =
    1.8110e+03
```

We can see that $v = 11.6133$ ft/sec and $q = 1811$ gpm.

2. The inside diameters of 4, 5, 6, and 8 in. schedule 40 steel pipe are 4.026, 5.047, 6.065, and 7.981 in., respectively. Thus, the row vector D representing the inside diameters can be set as $D = [4.026 5.047 6.065 7.981]$. The script *vwf* calculates flow velocities and flow rates for each D and L and generates plots showing v and q versus D and L .

```
% vwf.m: calculates flow velocities and flow rates
clear all; T = 60; rf = 0.00015; dz = 300; dP = -150;
D = [4.026 5.047 6.065 7.981]; nD = length(D);
L = 500:500:10000; nL = length(L); vr = []; v0 = 10; qr = [];
for i = 1:nD
    for j = 1:nL
        v(j) = fzero(@vwf, v0, [], T,L(j),D(i),rf,dz,dP);
        q(j) = (7.481*60)*(pi*v(j).* (D(i)/12).^2)/4; % gpm
    end
end
```

```

    vr = [vr v'];
    qr = [qr q'];
end
figure(1), plot(L,vr(:,1),'o',L,vr(:,2),'*',L,vr(:,3),'+',L,vr(:,4),'d')
legend('D=4in','D=5in','D=6in','D=8in'), axis([500 10000 0 20])
xlabel('L(ft)'), ylabel('Velocity, v(ft/s)')
figure(2), plot(L,qr(:,1),'o',L,qr(:,2),'*',L,qr(:,3),'+',L,qr(:,4),'d')
legend('D=4in','D=5in','D=6in','D=8in')
xlabel('L(ft)'), ylabel('Flow rate, q(gpm)')

```

Figure 5.9 shows plots of the flow velocity and flow rate versus pipe length and diameter produced by the script *vwfig*.

```
>> vwfig
```

3. Set $T = [40 60 100]$ and use the built-in function *fzero* to solve the nonlinear equation. The script *qrmT* performs calculation procedures.

```

% qrmT.m: calculates properties at various T
clear all;
T=[40 60 100]; nT = length(T); L=1000; D=7.981;
rf=0.00015; dz=300; dP=-150; v0 = 10;
rhor = []; mur = []; qr = [];
for i = 1:nT
    v = fzero(@vwfun, v0, [], T(i),L,D,rf,dz,dP);
    q = (7.481*60)*(pi*v.* (D/12).^2)/4;
    rho = 62.122+0.0122*T(i)-(1.54e-4)*T(i).^2+...
        (2.65e-7)*T(i).^3-(2.24e-10)*T(i).^4;
    mu = exp(-11.0318 + 1057.51./(T(i)+214.624));
    rhor = [rhor rho]; mur = [mur mu]; qr = [qr q];
end
qr, rhor, mur

```

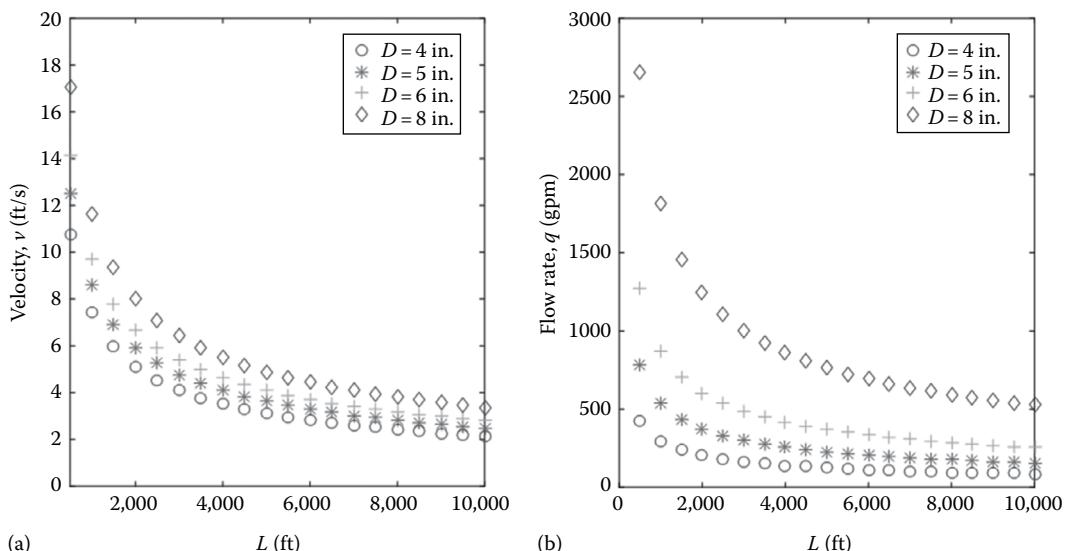


FIGURE 5.9 (a) Flow velocity and (b) flow rate versus pipe length and diameter.

TABLE 5.3
Calculation Results from Property Equations

Temperature (°F)	Density (lb _m /ft ³)	Viscosity (lb _m /(ft · sec))	Flow Rate (gpm)
40	62.3800	0.0010	1784.3
60	62.3539	0.0008	1811.0
100	62.0446	0.0005	1875.2

The script *qrmT* produces the following results, which are summarized in Table 5.3.

```
>> qrmT
qr =
1.0e+03 *
1.7843 1.8110 1.8752
rhor =
62.3800 62.3539 62.0446
musr =
0.0010 0.0008 0.0005
```

Example 5.9: Liquid Flow in a Pipeline

Water is drained from a reservoir through a pipeline from point 1, where the pressure is $P_1 = 205$ kPa, to point 2, where the pressure is $P_2 = 125$ kPa (see Figure 5.10). Calculate the mass flow rate in kg/sec for water required to generate 1.2 MW from the turbine installed between point 1 and point 2. The distance from the reservoir (point 1) to the turbine is 120 m and the distance from the turbine to point 2 is 5 m. Assume that temperature and fluid velocity are constant and the friction loss is negligible.

Solution

From assumptions of constant velocity and negligible friction loss, the momentum balance becomes

$$\frac{\Delta v^2}{2} + g\Delta z + \frac{\Delta P}{\rho} + F_f = g\Delta z + \frac{\Delta P}{\rho} = -\frac{\dot{W}_s}{\dot{m}}$$

This equation can be solved for the mass flow rate to give

$$\dot{m} = -\frac{\dot{W}_s}{g\Delta z + \frac{\Delta P}{\rho}}$$

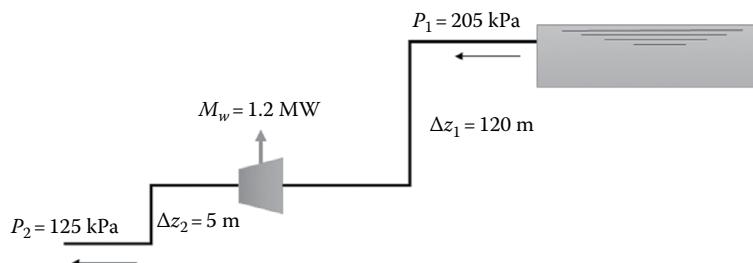


FIGURE 5.10 Flow in a pipeline with turbine.

The following commands produce the desired result:

```
>> P1=205; P2=125; dP=P2-P1; dz=-125; rho=1000; g=9.81; ws=1.2e6;
>> mdot = -ws/(dP/rho + g*dz)
mdot =
978.5294
```

5.3.2 EQUIVALENT LENGTH OF VARIOUS FITTINGS AND VALVES

The equivalent pipe length is a convenient method for determining the overall pressure drop ΔP in a pipe. The equivalent length is added to the length of actual straight pipe, L_{st} , to give the total length of pipe, L_{Total} :

$$L_{Total} = L_{st} + L_{eq}$$

The equations for the pressure drop ΔP and the head loss ΔH can be expressed as

$$\Delta P = \left(f_D \frac{L}{D} + \sum K \right) \frac{\rho v^2}{2g_c}, \quad \Delta H = \left(f_D \frac{L}{D} + \sum K \right) \frac{v^2}{2g_c}$$

where

D is the internal diameter of the pipe (ft)

f_D is the Darcy friction factor

K is the excess head loss

L is the pipe length (ft)

5.3.3 EXCESS HEAD LOSS

K is a dimensionless factor defined as the excess head loss in a pipe fitting and expressed in velocity heads. As a correlation for K , the 2- K method is known to be suitable for any pipe size.¹⁶ The 2- K method can be expressed as

$$K = \frac{K_1}{N_{Re}} + K_\infty \left(1 + \frac{1}{d} \right)$$

where

d is the internal diameter of attached pipe (in.)

K_1 is K for fitting at $N_{Re} = 1$

K_∞ is K for a large fitting at $N_{Re} = \infty$

The conversion between the equivalent pipe length and the resistance coefficient K can be represented as

$$K = f \frac{L_{eq}}{R_h} = 4f \frac{L_{eq}}{D} = f_D \frac{L_{eq}}{D}$$

Table 5.4 shows the values of K_1 and K_∞ for the 2- K method.

TABLE 5.4
Velocity Head Factors of Pipe Fittings

		Type of Fitting	K_1	K_∞		
Elbows	90°	Standard ($R/D = 1$), screwed	800	0.40		
		Standard ($R/D = 1$), flanged/welded	800	0.25		
		Long radius ($R/D = 1.5$), all types	800	0.20		
		Mitered elbows	1 Weld (90°)	1000	1.15	
			($R/D = 1.5$)	2 Weld (45°)	800	0.35
	45°			3 Weld (30°)	800	0.30
				4 Weld (22.5°)	800	0.27
				5 Weld (18°)	800	0.25
		Standard ($R/D = 1$), all types	500	0.20		
		Long radius ($R/D = 1.5$), all types	500	0.15		
Tees	Used as elbow	Mitered, 1 weld, 45°	500	0.25		
		Mitered, 2 weld, 22.5°	500	0.15		
		Standard ($R/D = 1$), screwed	1000	0.60		
		Standard ($R/D = 1$), flanged/welded	1000	0.35		
	180°	Long radius ($R/D = 1.5$), all types	1000	0.30		
Valves	Run-through tee	Standard, screwed	500	0.70		
		Long radius, screwed	800	0.40		
		Standard, flanged or welded	800	0.80		
		Stub-in-type branch	1000	1.00		
	Gate ball, plug	Screwed	200	0.10		
		Flanged or welded	150	0.05		
Globe, angle or Y type	Stub-in-type branch	Stub-in-type branch	100	0.00		
		Full line size, $\beta = 1.0$	300	0.10		
		Reduced trim, $\beta = 0.9$	500	0.15		
	Diaphragm, dam type	Reduced trim, $\beta = 0.8$	1000	0.25		
		Globe, standard	1500	4.00		
		Globe, angle or Y type	1000	2.00		
Butterfly	Check	Diaphragm, dam type	1000	2.00		
		Butterfly	800	0.25		
		Check	2000	10.00		
	Swing	Lift	1500	1.50		
		Swing	1000	0.50		
	Tilting disk	Tilting disk	1000	0.50		

Source: Coker, A.K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, 1995, p. 155.

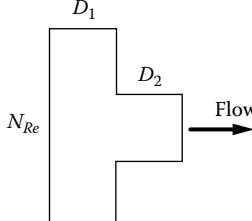
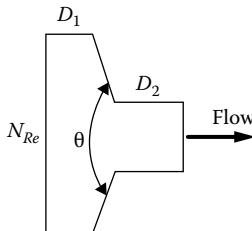
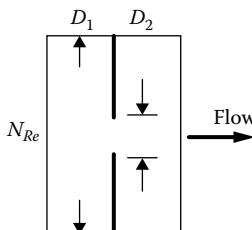
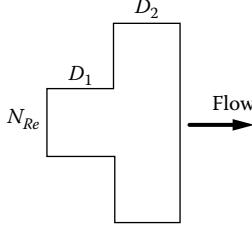
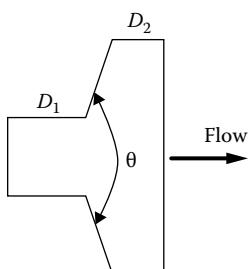
5.3.4 PIPE REDUCTION AND ENLARGEMENT

When the pipe size changes, the velocity head $v^2/2g_c$ also changes. The velocity is inversely proportional to the flow area and thus to the diameter squared. Therefore, K is inversely proportional to the velocity squared, and we can have the relation

$$K_2 = K_1 \left(\frac{D_2}{D_1} \right)^4$$

Table 5.5 shows how K varies with changes in pipe size.

TABLE 5.5
Correlation of K for Fitting Caused by a Change in Pipe Size

Fitting Type	Inlet N_{Re}	K Based on Inlet Velocity Head
	$N_{Re} \leq 2500$	$K = \left[1.2 + \frac{160}{N_{Re}} \right] \left[\left(\frac{D_1}{D_2} \right)^4 - 1 \right]$
	$N_{Re} > 2500$	$K = (0.6 + 0.48 f_D) \left(\frac{D_1}{D_2} \right)^2 \left[\left(\frac{D_1}{D_2} \right)^2 - 1 \right]$
1. Square reduction		
	All	Multiply K from Type 1 by $\sqrt{\sin(\theta/2)}$ for $45^\circ < \theta < 180^\circ$ or $1.6 \sin(\theta/2)$ for $0^\circ < \theta < 45^\circ$
2. Tapered reduction		
	$N_{Re} \leq 2500$	$K = \left[2.7 + \left(\frac{D_2}{D_1} \right)^2 \left(\frac{120}{N_{Re}} - 1 \right) \right] \left[1 - \left(\frac{D_2}{D_1} \right)^2 \right] \left[\left(\frac{D_1}{D_2} \right)^4 - 1 \right]$
	$N_{Re} > 2500$	$K = \left[2.7 - \left(\frac{D_2}{D_1} \right)^2 \left(\frac{4000}{N_{Re}} \right) \right] \left[1 - \left(\frac{D_2}{D_1} \right)^2 \right] \left[\left(\frac{D_1}{D_2} \right)^4 - 1 \right]$
3. Thin, sharp orifice		
	$N_{Re} \leq 4000$	$K = 2 \left[1 - \left(\frac{D_1}{D_2} \right)^4 \right]$
	$N_{Re} > 4000$	$K = (1 + 0.8 f_D) \left[1 - \left(\frac{D_1}{D_2} \right)^2 \right]^2$
Square expansion		
	All	If $\theta > 45^\circ$, use K from Type 4; otherwise, multiply K from Type 4 by $2.6 \sin(\theta/2)$
4. Tapered expansion		

5.3.5 OVERALL PRESSURE DROP

For steady-state flow, the Bernoulli equation yields¹⁹

$$\int_{P_1}^{P_2} \frac{dP}{\rho} + \frac{V_2^2 - V_1^2}{2g_c} + \frac{g}{g_c} (Z_2 - Z_1) + \Delta H = \hat{W}_s$$

where

$$\Delta H = \sum_{\text{all sections of straight pipe}} \frac{v^2}{2g_c} \left(4f \frac{L}{D} \right) + \sum_{\text{fitting, values, etc.}} K \frac{v^2}{2g_c}$$

and \hat{W}_s is the power required in the pump.

Sizing of process pipes often involves trial and error procedure. Usually, a pipe size is selected first and then the Reynolds number, friction factor, and coefficient of resistance are calculated. Then the pressure drop per 100 ft of pipe, ΔP_{100} , is computed. For a given volumetric flow rate and physical properties of a single-phase fluid, ΔP_{100} for laminar and turbulent flow is given by

$$\text{Laminar flow: } \Delta P_{100} = 0.0273 \frac{\mu Q}{d^4} \text{ (psi/100 ft)}$$

$$\text{Turbulent flow: } \Delta P_{100} = 0.0216 f_D \frac{\rho Q^2}{d^5} \text{ (psi/100 ft)}$$

where d is the internal diameter of the pipe (in.) and ρ is the density of the fluid (lb/ft³). If a mass flow rate and physical properties of a single-phase fluid are given, ΔP_{100} for laminar and turbulent flow is given by

$$\text{Laminar flow: } \Delta P_{100} = 0.0034 \frac{\mu W}{d^4 \rho} \text{ (psi/100 ft)}$$

$$\text{Turbulent flow: } \Delta P_{100} = 0.000336 f_D \frac{W^2}{d^5 \rho} \text{ (psi/100 ft)}$$

Multiplying ΔP_{100} by the total pipe length between two points and adding the pipe elevation yield the overall pressure drop ΔP (psi):

$$\Delta P = \Delta P_{100} \cdot \frac{L_{Total}}{100} + \frac{\rho \Delta z}{144} \text{ (psi)}$$

The pipe size for a given flow rate is often selected on the assumption that ΔP is close to or less than the available pressure difference between two points in the pipeline. If ΔP and mass flow rate W are given, the pipe diameter D can be determined from the relation

$$D = \left[\frac{2f(L + L_e)}{\rho \Delta P} \left(\frac{4W}{\pi} \right)^2 \right]^{1/5}$$

Example 5.10: Power Requirement from a Pump²⁰

Water is to be delivered to the upper tank as shown in Figure 5.11. Determine the required power output from the pump at steady state. All of the piping is 4 in. internal diameter smooth circular pipe.

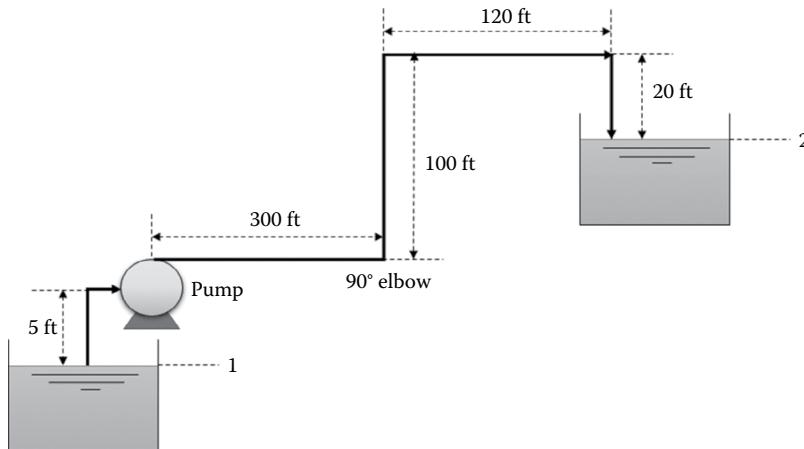


FIGURE 5.11 Pipeline flow with friction loss due to fittings. (From Bird, R.B. et al., *Transport Phenomena*, 2nd ed., John Wiley & Sons, Inc., Hoboken, NJ, 2002, pp. 207–208.)

Data: $\rho = 62.4 \text{ lb}_m/\text{ft}^3$, $\mu = 1 \text{ cp} = 0.0006905 \text{ lb}_m/\text{ft/sec}$, $Q = 12 \text{ ft}^3/\text{min} = 0.2 \text{ ft}^3/\text{sec}$, K (sudden contraction) = 0.45, K (sudden expansion) = 1, K (90° elbow) = 0.5, $\epsilon = 0.0005$, $g_c = 32.174 \text{ lb}_m \text{ ft}/(\text{lb}_f \text{ sec}^2)$.

Solution

The values of the pressure and the velocity at point 1 are equal to those at point 2, respectively. Thus, from the steady-state mechanical energy balance, we get

$$W_s = \frac{g}{g_c} (Z_2 - Z_1) + \Delta H, \quad \Delta H = \sum_{\text{all sections of straight pipe}} \frac{v^2}{2g_c} \left(4f \frac{L}{D} \right) + \sum_{\text{fitting, values, etc.}} K \frac{v^2}{2g_c}$$

The script *powfun* performs the required calculations.

```
% powfun.m
rho = 62.4; mu = 6.905e-4; D = 1/3; Q = 0.2; K = 0.45+3*0.5+1; gc = 32.174;
L = 5+300+100+120+20; eD = 5e-5;
v = 4*Q/(pi*D^2); m = Q*rho; Nre = D*v*rho/mu; % Reynolds 수
if Nre < 2100
    f = 16./Nre;
else % Shacham eqn
    den = 16*(log10(eD/3.7 - 5.02*log10(eD/3.7+14.5./Nre)).^2;
    f = 1./den;
end
dH = (4*f*L/D + K)*v^2/(2*gc);
Ws = m*(dH + (105-20));
v, f, Ws
```

The script *powfun* produces the following outputs. From the results, we can see that $W_s = 1096.8 \text{ ft}\cdot\text{lb/sec}$.

```
>> powfun
v =
2.2918
f =
0.0049
Ws =
1.0968e+03
```

Example 5.11: Internal Diameter of a Pipe

A process vapor is to be condensed with cooling water flowing inside the tubes of a shell-and-tube heat exchanger. The mass flow rate of cooling water is 0.36 kg/sec and the allowable pressure drop is 10 kPa. Determine the diameter of the tube to be used in the condenser. The friction factor in turbulent region may be given by the Shacham equation.

Data: $\rho = 85.9 \text{ kg/m}^3$, $\mu = 4.4 \times 10^{-4} \text{ N}\cdot\text{sec/m}^2$ (kg/m/sec), $L = 4 \text{ m}$, $\epsilon = 5 \times 10^{-6} \text{ m}$, $K = 0.3$.

Solution

Find the Reynolds number from the relation $v = 4W/\pi D^2\rho$. The resulting Reynolds number is used to calculate the friction factor. The solution of the nonlinear equation

$$f(D) = D - \left[\frac{2f(L+L_e)}{\rho\Delta P} \left(\frac{4W}{\pi} \right)^2 \right]^{1/5} = 0$$
 yields the diameter D (m). In this equation, L_e is defined by $L_e = \frac{KD}{4f}$. The function *dfunc* defines the nonlinear equation.

```
function fD = dfunc(D,W,rho,mu,rf,dP,L,K)
eD = rf./D; % roughness factor
v = 4*W./(pi*rho*D.^2); % m/s
Nre = D.*v.*rho./mu; % Reynolds number
if Nre < 2100
    f = 16./Nre;
else % Shacham eqn
    den = 16*(log10(eD/3.7 - 5.02*log10(eD/3.7+14.5./Nre))./Nre)).^2;
    f = 1./den;
end
Le = K*D/(4*f);
fD = D - ((2*f*(L + Le))./(rho*dP) * (4*W/pi)^2).^0.2;
end
```

The following commands set data and call the built-in function *fzero* to solve the nonlinear equation defined by the function *dfunc*:

```
>> W=0.36; rho=85.9; mu=4.4e-4; rf=5e-6; dP=1e4; L=4; K=0.3; D0=0.1;
>> D = fzero(@dfunc,D0, [], W, rho, mu, rf, dP, L, K)
D =
    0.0261
```

Example 5.12: Pipeline Flow²¹

A pipeline isometric (Figure 5.12) shows a 6 in. schedule 40 steel pipeline with six 90° LR (long radius) elbows and two flow-through tees. The actual length of the pipe is 78 ft. The mass flow rate of the fluid is 75,000 lb/hr, the viscosity and density of the fluid are 1.25 cP and 64.30 lb/ft³, respectively, and the pipe roughness is $\epsilon = 0.00015$ ft. Calculate the equivalent length of pipe fittings and total length of the pipe.

Solution

The script *eqplen* performs the required calculations. In the script, the matrix *cfdat* represents the values of K_1 and K_∞ for pipes with large radius.

```
% eqplen.m: calculate the equivalent length of pipe fittings and
% total length of the pipe
cfdat = zeros(14,3); cfdat(2,1) = 6; cfdat(11,1) = 2;
cfdat(:,2) = 100*[10 8 5 3 5 10 15 10 8 15 1.5 1 1.6 0]';
cfdat(:,3) = 0.1*[3 2 1.5 1 1.5 2.5 40 20 2.5 15 0.5 0 5 10]';
g = 32.2; d = 6.065; D = d/12; pr = 1.5e-4; w = 75000; mu = 1.25; rho = 64.3;
```

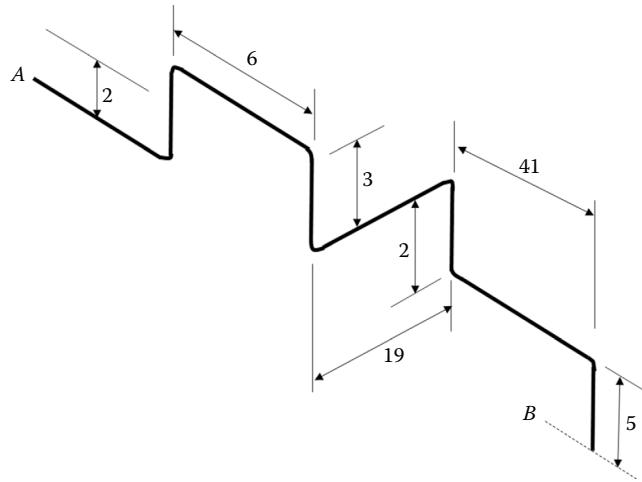


FIGURE 5.12 Pipeline isometric layout.

```

Lst = 78; z = 8; Area = pi*D^2/4; Nre = 6.31*w/(d*mu);
v = 0.0509*w/(rho*d^2); Q = w/(8.02*rho);
if Nre <= 2100 % laminar flow
    f = 64/Nre; % Darcy friction factor
else % turbulent: friction factor by Chen eqn.
    Av = pr/3.7/D + (6.7/Nre)^0.9;
    f = 4./(-4*log10(pr/D/3.7 - 5.02*log10(Av)/Nre)).^2;
end
Ksum = sum(cfdat(1:12,1).*cfdat(1:12,2));
Klsum = sum(cfdat(1:12,1).*cfdat(1:12,3));
Kt = Ksum/Nre + Klsum*(1 + 1/d);
Kee1 = sum(cfdat(13:14,1).*cfdat(13:14,2));
Kee2 = sum(cfdat(13:14,1).*cfdat(13:14,3));
K = Kee1/Nre + Kee2 + Kt; Kt = Kt + K + f*Lst/D;
Leq = K*D/f, L = Lst + Leq
if Nre <= 2100
    delP = 0.0034*mu*w/(d^4*rho);
else
    delP = 0.000336*f*w^2/(d^5*rho);
end
delPsi = delP*L/100 + z*rho/144;
delH = 0.000483*f*L*w^2/(d^5*rho^2) + z;

```

The script *eqplen* produces the following results:

```

>> eqplen
Leq =
    38.3099
L =
    116.3099

```

The equivalent length of pipe fittings, L_{eq} , is 38.31 ft, and total length of the pipe is $L = 116.31$ ft.

5.3.6 PIPELINE NETWORK

In the simulation of steady-state flows of a single fluid in a network of pipes, a large set of simultaneous equations has to be solved and often the calculations are complex and iterative. The fundamental equation to be satisfied is the mass flow conservation, which states that the flow into and out of each

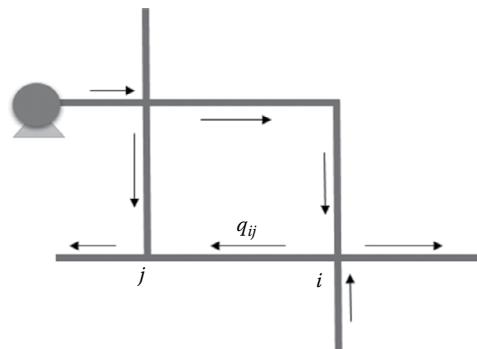


FIGURE 5.13 A simple pipeline network.

node should be equal. For a simple pipeline network shown in Figure 5.13, the pressure drop from node i to node j is given by

$$\Delta p_{ij} = \frac{32f\rho\Delta L_{i,j}}{\pi^2 D^5} q_{ij}^2$$

where $q_{i,j}$ represents the volumetric flow rate between node i and node j . The pressure drop equation states that the sum of pressure drops around any loop j should be zero:

$$\sum \xi_{ij} \Delta p_{ij} = 0$$

where ξ_{ij} is the direction of flow relative to flow loop j . Flows in clockwise direction (assumed flow direction) are treated positive and those in counterclockwise are treated negative.

Example 5.13: Flow in Pipeline Network²²

Water at 25°C is flowing in the pipeline network shown in Figure 5.14. The pressure at the exit of the pump is P_0 = 15 bar (gauge pressure), and the water is discharged at atmospheric pressure (P_5) at the end of the pipeline. All the pipes are 6 in. schedule 40 steel with an inside diameter of 0.154 m. The equivalent lengths of the pipes connecting different nodes are $L_{01} = 100$ m, $L_{12} = L_{23} = L_{45} = 300$ m, and $L_{13} = L_{24} = L_{34} = 1200$ m. Determine all the flow rates q_{ij} and pressures at nodes 1, 2, 3, and 4 for the pipeline network. The pipe roughness is $\epsilon = 4.6 \times 10^{-5}$ m and the friction factor f may be given by the Shacham equation. At 25°C, the viscosity of water is 8.931×10^{-4} kg/m/sec.

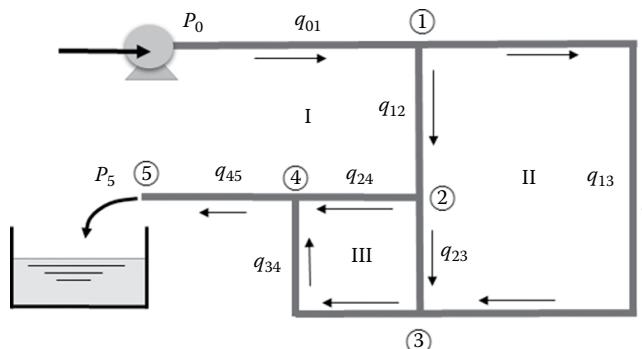


FIGURE 5.14 Pipeline network. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 309.)

Solution

Application of the conservation of mass balance yields

$$\text{Node ①: } q_{01} - q_{12} - q_{13} = 0$$

$$\text{Node ②: } q_{12} - q_{23} - q_{24} = 0$$

$$\text{Node ③: } q_{13} - q_{23} - q_{34} = 0$$

$$\text{Node ④: } q_{24} - q_{34} - q_{45} = 0$$

The summation of pressure drops around various loops gives

$$\text{Loop I: } \Delta p_{01} + \Delta p_{12} + \Delta p_{24} + \Delta p_{45} + \Delta p_{pump} = 0$$

$$\text{Loop II: } \Delta p_{13} - \Delta p_{12} - \Delta p_{23} = 0$$

$$\text{Loop III: } \Delta p_{23} - \Delta p_{24} - \Delta p_{34} = 0$$

where $\Delta p_{pump} = 0 - 15 = -15$ bar = -15×10^5 Pa. The MATLAB function *pnetq* defines the system of nonlinear equations. In this function, $x(k)$ denotes q_{ij} .

```
function fun = pnetq(x,D,rho,mu,dP0,L)
% x1=q01, x2=q12, x3=q13, x4=q23, x5=q24, x6=q34, x7=q45
% L1=L01, L2=L12, L3=L13, L4=L23, L5=L24, L6=L34, L7=L45

A = pi*D.^2/4; eD = 4.6e-5./D;
for k = 1:7
    Nre = D*x(k)*rho/mu/A; % Shacham eqn.
    f = 1./((log10(eD/3.7 - (5.02./Nre).*log10(eD/3.7+14.5/Nre))).^2/16;
    dP(k) = 32*f*rho*L(k)*(x(k)).^2/(pi.^2*D.^5);
end
fun = [x(1) - x(2) - x(3);
        x(2) - x(4) - x(5);
        x(3) + x(4) - x(6);
        x(5) + x(6) - x(7);
        dP(1) + dP(2) + dP(5) + dP(7) + dP0;
        -dP(2) + dP(3) - dP(4);
        dP(4) - dP(5) + dP(6)];
end
```

The initial values for x are all set to 0.1. The nonlinear equation system defined by the function *pnetq* may be solved by using the built-in function *fsolve*.

```
>> L = 100*[1 3 12 3 12 12 3]; dP0 = -15e5; rho = 997.08; mu = 8.931e-4;
>> D = 0.154; x0 = 0.1*ones(1,7);
>> x = fsolve(@pnetq,x0,[],D,rho,mu,dP0,L)
x =
    0.1098 0.0730 0.0368 0.0177 0.0553 0.0545 0.1098
```

We can see that $q_{01} = 0.1098$, $q_{12} = 0.073$, $q_{13} = 0.0368$, $q_{23} = 0.0177$, $q_{24} = 0.0553$, $q_{34} = 0.0545$, $q_{45} = 0.1098$ (m^3/sec).

The Lang–Miller method²³ to solve pipe network problems uses the generalized friction factor correlation of Churchill that spans the full range of the Reynolds numbers. The pressure drop is expressed as²⁴

$$\Delta P = c f \dot{m}^2, \quad c = \frac{32L}{\rho D_i^5 \pi^2}$$

where

c is a constant for a given fluid and pipe

the subscript i denotes a loop in the network

L and D are the length and the internal diameter of pipelines in the corresponding loop, respectively

The basic assumption in the Lang–Miller method is that the friction factor can be empirically represented as

$$f = a\dot{m}_i^{-b}$$

where a and b are both assumed constants over the range of the Reynolds numbers for one single iteration. Using this relation, we have

$$\Delta P_i = c\dot{m}^2 = a\dot{m}_i^{2-b} = \phi\dot{m}_i^{2-b}$$

For each loop in the pipeline network, we can get the following equations:

$$f = 2 \left[\left(\frac{8}{N_{Re}} \right)^{12} + \frac{1}{(A+B)^{1.5}} \right]^{\frac{1}{12}}, \quad A = (2.457 \ln C)^{16}, \quad B = \left(\frac{37,530}{N_{Re}} \right)^{16}, \quad C = \left(\frac{7}{N_{Re}} \right)^{0.9} + 0.27 \frac{\epsilon}{D}$$

$$b = \frac{N_{Re}}{8} \left[\left[\frac{8}{N_{Re}} \right]^{13} + \frac{1}{(A+B)^{2.5}} \left(\frac{dA}{dN_{Re}} + \frac{dB}{dN_{Re}} \right) \right] \frac{1}{(f/2)^{12}}$$

$$\frac{dA}{dN_{Re}} = \frac{39.312}{C} (2.457 \ln C)^{15} \frac{dC}{dN_{Re}}, \quad \frac{dB}{dN_{Re}} = -\frac{16B}{N_{Re}}, \quad \frac{dC}{dN_{Re}} = -0.9 \left(\frac{7}{N_{Re}} \right)^{0.9} \frac{1}{N_{Re}}, \quad a = f\dot{m}^b$$

Substitution of the equation for pressure drops to the summation relation yields

$$\sum_i \xi_{ij} \phi_i \dot{m}_i^{2-b_i} = 0 \quad (j=1,2,\dots,n)$$

where n is the number of loops.

Example 5.14: Four-Loop Network²⁵

A pipeline network with four flow loops is illustrated in Figure 5.15. The lengths and internal diameters of the pipes are shown in Table 5.6. Calculate the mass flow rate in each pipe that satisfies the pressure drop equation. A pipe roughness of $\epsilon = 5 \times 10^{-6}$ m is assumed uniformly for all pipes. The density and viscosity of the fluid are 881 kg/m^3 and $5 \times 10^{-4} \text{ N}\cdot\text{sec}/\text{m}^2$, respectively, and the inlet flow rate is $\dot{m}_0 = 334 \text{ kg/sec}$.

Solution

Application of the conservation of mass balance at each junction yields the following nine equations:

$$m_1 + m_4 - 334 = 0, \quad m_1 - m_2 - m_5 = 0, \quad m_3 - m_4 + m_8 = 0,$$

$$m_2 + m_3 - m_7 - m_9 = 0, \quad m_5 - m_6 - 42 = 0, \quad m_6 + m_7 - m_{12} - 108 = 0,$$

$$m_{11} + m_{12} - 9 = 0, \quad m_9 + m_{10} - m_{11} - 88 = 0, \quad m_8 - m_{10} - 87 = 0$$

From the fact that the sum of pressure drops at each loop is equal to zero, we can get three equations:

$$\phi_1 m_1^{2-b_1} + \phi_2 m_2^{2-b_2} - \phi_3 m_3^{2-b_3} - \phi_4 m_4^{2-b_4} = 0$$

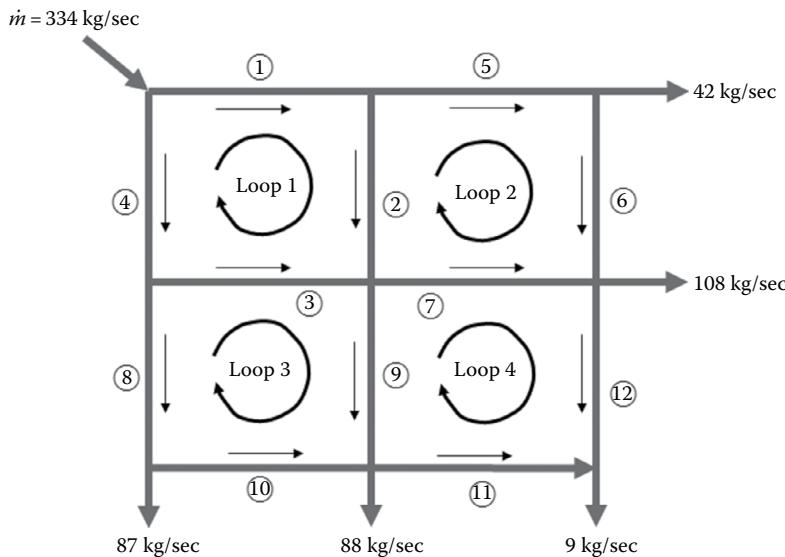


FIGURE 5.15 Four-loop pipeline network. (From Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, 1985, pp. 234–235.)

TABLE 5.6
Lengths and Internal Diameters of the Pipes

Flow (i)	Diameter (D_i , m)	Length (L_i , m)	Flow (i)	Diameter (D_i , m)	Length (L_i , m)
1	0.508	915	7	0.305	915
2	0.406	1220	8	0.406	1220
3	0.406	915	9	0.305	1220
4	0.610	1220	10	0.406	915
5	0.508	915	11	0.305	915
6	0.406	1220	12	0.305	1220

Source: Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, 1985, pp. 234–235.

$$-\phi_2 m_2^{2-b_2} + \phi_5 m_5^{2-b_5} + \phi_6 m_6^{2-b_6} - \phi_7 m_7^{2-b_7} = 0$$

$$\phi_3 m_3^{2-b_3} - \phi_8 m_8^{2-b_8} + \phi_9 m_9^{2-b_9} - \phi_{10} m_{10}^{2-b_{10}} = 0$$

Now we have 12 nonlinear equations to be solved. The nonlinear equation system is defined by the MATLAB function *pipnet*.

```
function fm = pipnet(m, rho, mu, D, L, rf)
v = 4*m.^(pi*D.^2*rho); Nre = D.*v*rho/mu;
C = (7./Nre).^0.9 + 0.27*rf./D;
A = (2.457*log(C)).^16; B = (37530./Nre).^16;
f = 2*((8./Nre).^12 + 1./((A+B).^1.5)).^(1/12);
dC = -(0.9./Nre).*(7./Nre).^0.9;
dA = (39.312./C).*dC.* (2.457*log(C)).^15;
dB = -16*B./Nre;
b = (Nre/8)./((f/2).^12).*((8./Nre).^13 + (dA+dB)./((A+B).^2.5));
c = 32.*L./((rho*pi.^2*D.^5));
```

```

a = f.*m.^b; phi = a.*c;
fm = [m(1) + m(4) - 334;
m(1) - m(2) - m(5);
m(3) - m(4) + m(8);
m(2) + m(3) - m(7) - m(9);
m(5) - m(6) - 42;
m(6) + m(7) - m(12) - 108;
m(11) + m(12) - 9;
m(9) + m(10) - m(11) - 88;
m(8) - m(10) - 87;
phi(1)*m(1)^(2-b(1)) + phi(2)*m(2)^(2-b(2)) - phi(3)*m(3)^(2-b(3)) -
phi(4)*m(4)^(2-b(4));
-phi(2)*m(2)^(2-b(2)) + phi(5)*m(5)^(2-b(5)) + phi(6)*m(6)^(2-b(6)) -
phi(7)*m(7)^(2-b(7));
phi(3)*m(3)^(2-b(3)) - phi(8)*m(8)^(2-b(8)) + phi(9)*m(9)^(2-b(9)) -
phi(10)*m(10)^(2-b(10));
end

```

The script file *usepipnet* defines data, calls the function *pipnet*, and solves the nonlinear system using the built-in function *fsolve*.

```

% usepipnet.m
rho = 881; mu = 5e-4;
D = [0.508 0.406 0.406 0.610 0.508 0.406 0.305 0.406...
      0.305 0.406 0.305 0.305];
L = [915 1220 915 1220 915 1220 915 1220 915 915 1220];
rf = 5e-6*ones(1,length(D)); m0 = 20*ones(1,length(D));
m = fsolve(@pipnet,m0,[],rho,mu,D,L,rf)

```

The script *usepipnet* produces the following outputs:

```

>> usepipnet
Equation solved.
fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.
<stopping criteria details>
m =
153.8314 36.6770 61.6102 180.1686 117.1544 75.1544 47.6151 118.5584
50.6722 31.5584 -5.7694 14.7694

```

The negative value implies that the flow direction is opposite to the assumed direction indicated in Figure 5.15.

5.4 FLOW THROUGH TANK

5.4.1 OPEN TANK

Figure 5.16 shows an open tank flow system. The change of the liquid level with respect to time is given by

$$A \frac{dz}{dt} = F_1 - F_2$$

where A is the cross-sectional area of the tank. Each flow rate is a function of pressure difference and is given by

$$F_1 = C_{v1}\sqrt{P_1 - P_2}, \quad F_2 = C_{v2}\sqrt{P_2 - P_3}, \quad P_2 = P_0 + \rho g z$$

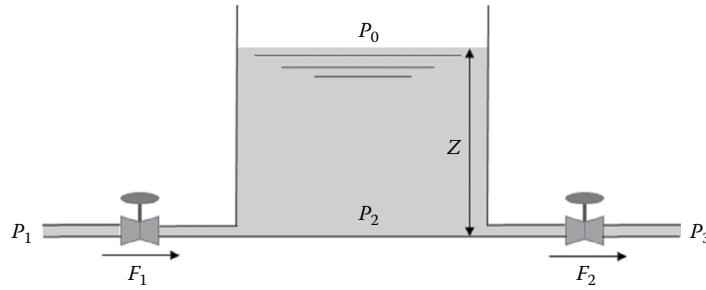


FIGURE 5.16 Open tank flow system. (From Ramirez, W.F., *Computational Methods for Process Simulation*, 2nd ed., Butterworth Heinemann, Oxford, UK, 1997, p. 145.)

Now we introduce dimensionless variables and parameters as

$$z^* = \frac{z}{z_0}, \quad t^* = \frac{t}{\tau}, \quad F^* = \frac{F}{F_C}, \quad P^* = \frac{P}{P_{10}}, \quad \tau = \frac{Az_0}{F_C}, \quad F_C = C_{v1}\sqrt{P_{10} - P_{20}}$$

Then, because $dz = z_0 dz^*$, $dt = \tau dt^*$, and $F_i = F_c F^*$ ($i = 1, 2$), we get the following dimensionless balance equations:

$$\begin{aligned} \frac{dz^*}{dt^*} &= F_1^* - F_2^*, \quad F_1^* = \sqrt{\frac{P_{10}}{P_{10} - P_{20}}} \sqrt{P_1^* - P_2^*}, \quad F_2^* = \left(\frac{C_{v2}}{C_{v1}} \right) \sqrt{\frac{P_{10}}{P_{10} - P_{20}}} \sqrt{P_2^* - P_3^*} \\ P_2^* &= P_0^* + \left(\frac{\rho g z_0}{P_{10}} \right) z^* \end{aligned}$$

5.4.2 ENCLOSED TANK

For an enclosed tank as shown in Figure 5.17, the pressure $P_0 = P_G$ above the liquid surface is not constant anymore. The movement of the liquid surface up and down will compress and expand the gas and cause the pressure to change. Assuming an ideal gas, we can use the equation of state

$$P_G = \frac{nRT_G}{V_G}$$

where V_G and T_G are the gas volume and temperature, respectively. Under the assumption of adiabatic conditions, $c_v dT_G = -(P_G/m) dV_G$, and we can get

$$\frac{T_G}{T_{G0}} = \left(\frac{V_{G0}}{V_G} \right)^{R/(c_v M_w)}$$

where M_w is the molecular weight of gas (air). The gas volume is given by

$$V_G = V_0 - Az$$

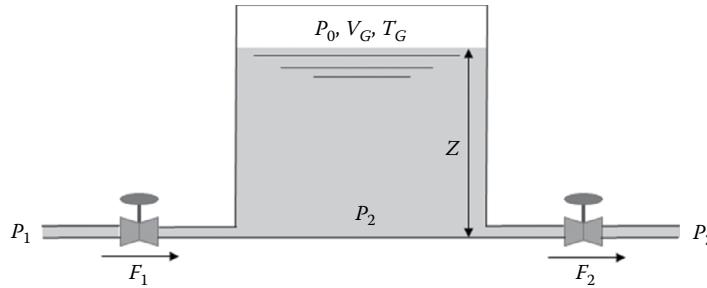


FIGURE 5.17 Enclosed tank. (From Ramirez, W.F., *Computational Methods for Process Simulation*, 2nd ed., Butterworth Heinemann, Oxford, UK, 1997, p. 145.)

We define dimensionless variables and parameters as

$$z^* = \frac{z}{z_0}, \quad t^* = \frac{t}{\tau}, \quad F^* = \frac{F}{F_C}, \quad P^* = \frac{P}{P_{10}}, \quad V_G^* = \frac{V_G}{V_{G0}}, \quad T_G^* = \frac{T_G}{T_{G0}}, \quad \tau = \frac{Az_0}{F_C}, \quad F_C = C_{v1}\sqrt{P_{10} - P_{20}}$$

Then, from the relations $dz = z_0 dz^*$, $dt = \tau dt^*$, $F_i = F_C F^*$ ($i = 1, 2$), $V_G = V_{G0} V_G^*$, and $T_G = T_{G0} T_G^*$, we have the following equations:

$$\begin{aligned} \frac{dz^*}{dt^*} &= F_1^* - F_2^*, \quad F_1^* = \sqrt{\frac{P_{10}}{P_{10} - P_{20}}} \sqrt{P_1^* - P_2^*}, \quad F_2^* = \left(\frac{C_{v2}}{C_{v1}} \right) \sqrt{\frac{P_{10}}{P_{10} - P_{20}}} \sqrt{P_2^* - P_3^*} \\ P_G^* &= \left(\frac{P_{G0}}{P_{10}} \right) \frac{T_G^*}{V_G^*}, \quad P_2^* = P_G^* + \left(\frac{\rho g z_0}{P_{10}} \right) z^*, \quad V_G^* = \frac{V_0}{V_{G0}} - \frac{Az_0}{V_{G0}} z^*, \quad T_G^* = \left(\frac{1}{V_G^*} \right)^{R/(c_v M_w)} \end{aligned}$$

Example 5.15: Flow through Enclosed Tank²⁶

For a flow involving an enclosed tank, plot the dimensionless level (z^*), flow rates (F_1^* , F_2^*), and pressure (P_2^*) versus the dimensionless time (t^*) within the range of $0 \leq t^* \leq 0.4$. Assume that the molecular weight of the gas is 29 (air) and $P_{G0} = 0.735 P_{10}$.

Data:

$$\rho = 1000 \text{ kg/m}^3, \quad A = 0.465 \text{ m}^2, \quad z_0 = 3.05 \text{ m}, \quad g = 9.8 \text{ m/sec}^2, \quad C_{v1} = C_{v2} = 0.012 \text{ m}^2/\text{N}^{1/2}/\text{sec}$$

$$P_{10} = P_1 = 1.38 \times 10^5 \text{ N/m}^2, \quad P_{30} = P_3 = 1.08 \times 10^5 \text{ N/m}^2, \quad P_0 = 1.014 \times 10^5 \text{ N/m}^2$$

$$V_0 = 2.83 \text{ m}^3, \quad V_{G0} = 1.415 \text{ m}^3, \quad T_{G0} = 338.6 \text{ K}, \quad c_v = 0.2 \text{ cal/g/K}, \quad R = 1.987 \text{ cal/g mol/K}$$

Solution

Differential equations are defined by the MATLAB function *cltank*. The script *usecltank* calls the function *cltank* and generates the graph.

```
cltank.m
function dz = cltank(t, z, k1, k2, k3, k4, k5, k6, P10, P1, P3)
P1s = P1/P10; P3s = P3/P10; Vg = k4 - z; Tg = (1/Vg)^k6;
Pg = k5*Tg/Vg; P2s = Pg + k3*z;
```

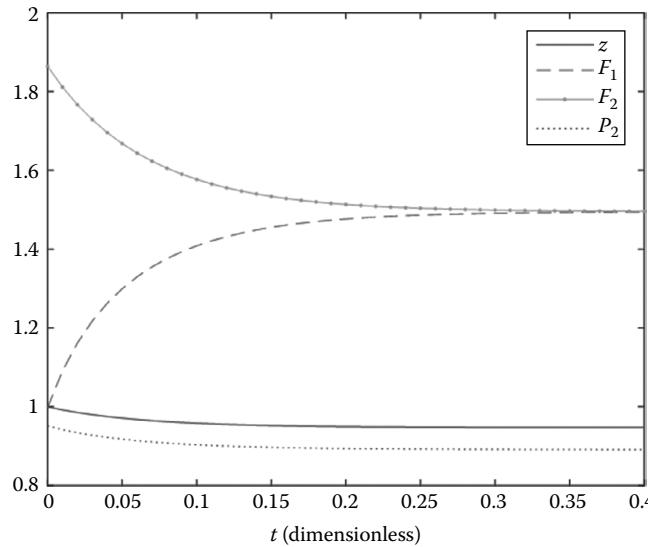


FIGURE 5.18 Dimensionless level, flow rate, and pressure versus time.

```

F1s = k1*sqrt(P1s - P2s); F2s = k2*sqrt(P2s - P3s);
dz = F1s - F2s;
end

usecltank.m
% usecltank.m
Cv1 = 1.2e-2; Cv2 = Cv1; P0 = 1.014e5; P10 = 1.38e5; P30 = 1.08e5;
P1 = P10; P3 = P30; z0 = 3.05; rho = 1e3; A = 0.465; g = 9.8;
P20 = P0 + rho*g*z0;
cv = 0.2; R = 1.987; Mw = 29; V0 = 2.83; Vg0 = 1.415;
k1 = sqrt(P10/(P10-P20)); k2 = k1*Cv2/Cv1;
k3 = rho*g*z0/P10; k4 = V0/Vg0; k5 = 0.735; k6 = R/(cv*Mw);
tspan = [0 0.4]; z0 = 1;
[t z] = ode45(@cltank,tspan,z0,[],k1,k2,k3,k4,k5,k6,P10,P1,P3);
Vg = k4 - z; Tg = (1./Vg).^k6; Pg = k5*Tg./Vg;
P1s = P1/P10; P3s = P3/P10; P2s = Pg + k3*z;
F1s = k1*sqrt(P1s - P2s); F2s = k2*sqrt(P2s - P3s);
plot(t,z,t,F1s,'--',t,F2s,'.-',t,P2s,':')
legend('z','F_1','F_2','P_2'), xlabel('t(dimensionless)')

```

The script *usecltank* produces the plot shown in Figure 5.18:

```
>> usecltank
```

5.5 FLOW OF NON-NEWTONIAN FLUID

5.5.1 VELOCITY PROFILE

For non-Newtonian fluid with power law characteristics, the shear stress can be described by

$$\tau_{rx} = -k \left| \frac{dv_x}{dr} \right|^{n-1} \frac{dv_x}{dr}$$

where k ($\text{N}\cdot\text{sec}^n/\text{m}^2$) is a parameter and n is the flow behavior index. When $n = 1$, the fluid is Newtonian. For $n > 1$, the fluid is dilatant, and for $n < 1$, the fluid is pseudoplastic. The velocity profile for a non-Newtonian fluid flowing in a horizontal circular pipe is given by

$$v_x = \frac{n}{n+1} \left(\frac{\Delta P}{2kL} \right)^{1/n} R^{(n+1)/n} \left[1 - \left(\frac{r}{R} \right)^{(n+1)/n} \right]$$

Integration of the velocity profile followed by division by cross-sectional area yields the average velocity as

$$v_{x,avg} = \frac{1}{\pi R^2} \int_0^R 2\pi r v_x dr = \frac{n}{3n+1} \left(\frac{\Delta P}{2kL} \right)^{1/n} R^{(n+1)/n}$$

The MATLAB function *nnhzpipe* calculates the average velocity and the shear stress and generates a plot of velocity profile for laminar flow of a non-Newtonian fluid in a horizontal pipe.

```
function [avgv taurx] = nnhzpipe(L,R,K,n,delP)
% average velocity and velocity profile for a non-Newtonian fluid flowing
% in a horizontal pipe
% input
% L: pipe length (m)
% R: pipe diameter (m)
% K: parameter (N*s^n/m^2)
% n: flow index
% delP: pressure drop (Pa)
% output
% avgv: avg. velocity (m/s)

h = 2*R/100; r = [-R:h:R]; % h: step size
Rn = R.^((n+1)/n); Pn = (delP./(2*K*L)).^(1/n);
v = (1 - (abs(r)/R).^(n+1/n)) .* Pn .* Rn * n/(n+1);
avgv = Rn .* Pn .* n/(3*n+1);
dvr = diff(v)/h; dvr = [dvr dvr(end)]; taurx = -K.*abs(dvr).^(n-1).*dvr;
subplot(1,2,1), plot(v,r), grid, xlabel('v_x(r)'), ylabel('r'), axis tight
subplot(1,2,2), plot(taurx,r), grid, xlabel('\tau_{rx}(r)'), ylabel('r'), axis tight
end
```

Example 5.16: Flow of a Non-Newtonian Fluid in a Horizontal Pipe

A non-Newtonian fluid is flowing in a horizontal pipe where $L = 15$ m, $r = 0.009$ m, $n = 2$, $k = 1.0 \times 10^{-6}$ $\text{N}\cdot\text{sec}^n/\text{m}^2$, and $\Delta P = 110$ Pa. Calculate the average velocity, and plot the velocity profile and shear stress versus radius.

Solution

The function *nnhzpipe* calculates the average velocity and generates the plots shown in Figure 5.19:

```
>> L=15; R=0.009; K=1e-6; n=2; delP=110;
>> avgv = nnhzpipe(L,R,K,n,delP)
avgv =
0.4671
```

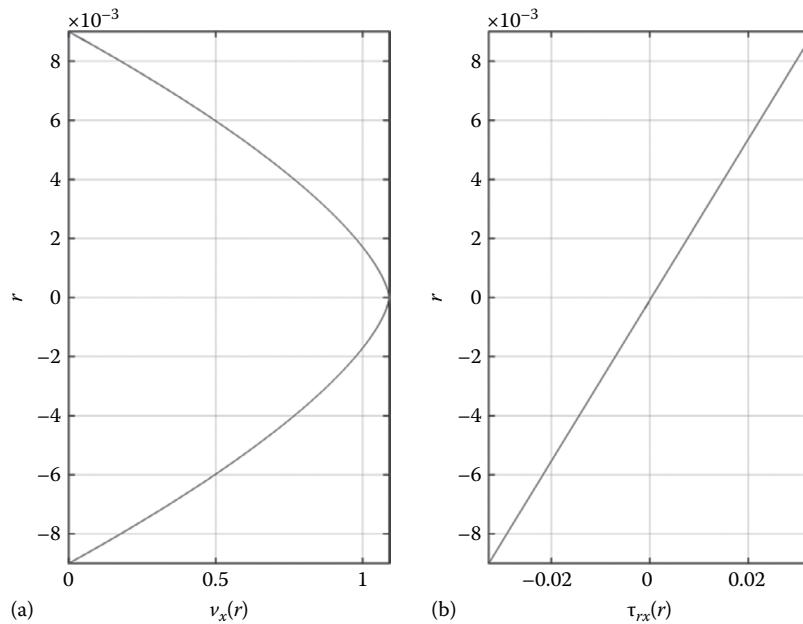


FIGURE 5.19 (a) Velocity profile and (b) shear stress for a non-Newtonian fluid flowing in a horizontal pipe.

5.5.2 REYNOLDS NUMBER

The constitutive equation for pipe flow of a Bingham plastic fluid is given by

$$\tau_{rz} = \tau_y + \eta \left(-\frac{du}{dr} \right)$$

where

u is the velocity

r and z are the radial and axial coordinates, respectively

The modified Reynolds number is defined as

$$N_{ReB} = \frac{Dup}{\eta}$$

The Hedstrom number is defined as

$$N_{He} = \frac{\tau_y D^2 \rho}{\eta^2}$$

Using N_{He} , the modified Reynolds number can be expressed as

$$N_{ReB} = \frac{N_{He}}{8\xi} \left(1 - \frac{4}{3}\xi + \frac{1}{3}\xi^4 \right)$$

where $\xi = \tau_y / \tau_w$.

For the power law fluid, the constitutive equation for one-dimensional pipe flow of a Bingham plastic fluid is given by

$$\tau_{rx} = -k \left| \frac{du}{dr} \right|^{n-1} \frac{du}{dr}$$

The Reynolds number for pipe flow of power law fluids (density: ρ) is given by²⁷

$$N_{Re} = \frac{D^n u^{2-n} \rho}{8^{n-1} k \left(\frac{3n+1}{4n} \right)^n}$$

where

D is the pipe diameter

u is the velocity

The generalized Reynolds number can be defined as

$$N_{Reg} = \frac{D^{n'} u^{2-n'} \rho}{8^{n'-1} k'}$$

For Bingham plastic fluids,

$$n' = \frac{\xi^4 - 4\xi + 3}{3(1 - \xi^4)}, \quad k' = \frac{\tau_y}{\xi} \left[\frac{3\eta\xi}{\tau_y (\xi^4 - 4\xi + 3)} \right]^{n'}$$

For power law fluids,

$$n' = n, \quad k' = k \left(\frac{3n+1}{4n} \right)^n$$

5.5.3 FRICTION FACTOR

For Bingham plastic fluid, the relationship between the Fanning friction factor and the Reynolds number can be expressed as²⁸

$$\text{Laminar region } (N_{ReB} \leq 2100): \frac{1}{N_{ReB}} = \frac{f}{16} - \frac{1}{6} \frac{N_{He}}{N_{ReB}^2} + \frac{\frac{1}{3} N_{He}^4}{f^3 N_{ReB}^8}$$

$$\text{Turbulent region } ^{29}: \frac{1}{\sqrt{f}} = 4 \log \left(N_{ReB} \phi(\xi) \sqrt{f} \right) - 0.4$$

where N_{He} is the Hedstrom number and ϕ is defined as

$$\phi(\xi) = \left(1 - \xi \right) \left(1 - \frac{4}{3} \xi + \frac{1}{3} \xi^4 \right)$$

$\xi = \tau_y/\tau_w$ is obtained from the solution of the equation

$$N_{ReB} - \frac{N_{He}}{8\xi} \left(1 - \frac{4}{3}\xi + \frac{1}{3}\xi^4 \right) = 0$$

For power law fluids, the relationship between the Fanning friction factor and the Reynolds number can be expressed as

$$\begin{aligned} \text{Laminar region } (N_{Re} \leq 2100): f &= 16/N_{Re} \\ \text{Turbulent region}^{30}: \frac{1}{\sqrt{f}} &= \frac{4}{n^{0.75}} \log(N_{Re} f^{1-n/2}) - \frac{0.4}{n^{1.2}} \end{aligned}$$

If we use the generalized Reynolds number N_{Reg} , the relationship between the Fanning friction factor and the Reynolds number can be expressed as³¹

$$\begin{aligned} \text{Laminar region } (N_{Reg} \leq 2100): f &= \frac{16}{N_{Reg}} \\ \text{Turbulent region: } \frac{1}{\sqrt{f}} &= \frac{4}{n^{0.75}} \log(N_{Reg} f^{1-n/2}) - \frac{0.4}{n^{1.2}} \end{aligned}$$

Example 5.17: Pipe Diameter for Non-Newtonian Flow

Calculate the pipe diameter required for the horizontal flow of a power law fluid in a pipe.³²

Data: Density $\rho = 961 \text{ kg/m}^3$, mass flow rate $\dot{m} = 6.67 \text{ kg/sec}$, pipe length $L = 10 \text{ m}$, pipe roughness $\epsilon = 5 \times 10^{-6} \text{ m}$, pressure drop $\Delta = 15 \text{ kPa}$, $K = 1.8$, $n = 0.64$, $k = 1.48 \text{ N} \cdot \text{sec}^{2-n}/\text{m}^2$.

Solution

The Reynolds number and the friction factor are calculated first and the diameter D is determined by solving the nonlinear equation

$$f(D) = D - \left[\frac{2f(L + L_e)}{\rho \Delta P} \left(\frac{4W}{\pi} \right)^2 \right]^{1/5} = 0$$

where $L_e = KD/4f$. The calculation procedure is performed by the function *nnDfun*.

```
function fD = nnDfun(D,W,rho,dP,rf,L,k,n,K)
eD = rf./D; % roughness
v = 4*W./(pi*rho*D.^2); % m/s
Nre = D.^n.*v.^2.*rho/(8^(n-1).*k* ((3*n+1)/(4*n)).^n); % Reynolds number
if Nre < 2100
    f = 16/Nre;
else
    f0 = 16/Nre;
    fe = @(x) 4*log10(Nre*x^(1-n/2))/n^0.75 - 0.4*n^(-1.2) - 1/sqrt(x);
    f = fzero(fe,f0);
end
Le = K*D/(4*f);
fD = D - ((2*f*(L + Le))./(rho*dP) * (4*W/pi)^2).^0.2;
end
```

The following commands produce the desired result. As an initial guess for D , set $D0 = 0.1$.

```
>> W=6.67; rho=961; dP=1.5e4; rf=5e-6; L=10; k=1.48; n=0.64; K=1.8;
>> D0 = 0.1; D = fzero(@nnDfun,D0, [], W, rho, dP, rf, L, k, n, K)
D =
0.0888
```

5.6 COMPRESSIBLE FLUID FLOW IN PIPES

5.6.1 CRITICAL FLOW AND THE MACH NUMBER

The flow rate of a compressible fluid in a pipe with a given upstream pressure approaches a certain maximum value that it cannot exceed even with reduced downstream pressure. The maximum velocity that a compressible fluid can attain in a pipe is known as the sonic velocity, V_s , and can be expressed as

$$V_s = 223 \sqrt{\frac{kT}{M_w}} = 68.1 \sqrt{\frac{kP_1}{\rho_1}} \quad (\text{ft/sec})$$

where the subscript 1 denotes upstream. At the sonic velocity, a critical pressure, P_c , is attained. If P_c is less than terminal pressure P_2 , the flow is subcritical. If P_c is greater than P_2 , then the flow is critical. The critical pressure can be found from the Crocker equation:

$$P_c = \frac{G}{11,400d^2} \sqrt{\frac{RT}{k(k+1)}} \quad (\text{psia})$$

where R is the molar gas constant given by $R = 1544S_g/29$ and the specific gravity S_g is defined by $S_g = (\text{molecular weight of the gas})/(\text{molecular weight of air})$. The upstream fluid velocity is determined from the relation $V = 0.0509G/d^2\rho_1$ (ft/sec). A recommended compressible fluid velocity for trouble-free operation is $V \leq 0.6V_s$.³³ Table 5.7 shows the design criteria for carbon steel vapor lines.

The Mach number, M , is the velocity of the gas divided by the velocity of the sound in the gas and can be represented as $M = V/V_s$. The exit Mach number for compressible isothermal fluid has been shown to be $1/\sqrt{k}$ where k is the ratio of the fluid specific heat capacities at constant pressure to that at constant volume (i.e., $k = C_p/C_v$). The flow is subsonic for $M < 1/\sqrt{k}$, sonic for $M = 1/\sqrt{k}$, and supersonic for $M > 1/\sqrt{k}$. Table 5.8 shows the k values for some common gases.

5.6.2 COMPRESSIBLE ISOTHERMAL FLOW

Estimation of the maximum flow rate and pressure drop of compressible isothermal flow is based on the following assumptions:

1. The fluid is isothermal compressible fluid.
2. The gas obeys the ideal gas law.
3. The friction factor is constant along the pipe.
4. No mechanical work is done on or by the system.
5. Steady-state flow is maintained.

Figure 5.20 shows the distribution of fluid energy with work done by the pump and heat added to the system.

The Bernoulli equation for the steady flow of a fluid can be written as

$$\int_1^2 \frac{dP}{\rho} + \frac{V^2}{2g_c\alpha} + \frac{g}{g_c} \Delta Z + h_L + \delta W_s = 0$$

where α is the dimensionless velocity distribution and $\alpha = 1$ for turbulent or plug flow. Since no mechanical work is done on the system, $\delta W_s = 0$, and $Z_2 = Z_1$ for horizontal pipe. The velocity head

TABLE 5.7
Recommended Velocity and Maximum ΔP for Carbon Steel Vapor Lines

Type		Recommended Velocity (ft/sec)	Maximum ΔP (psi/100 ft)
Fluid pressure (psig)	Subatmospheric		0.18
	$0 < P \leq 50$		0.15
	$50 < P \leq 150$		0.30
	$150 < P \leq 200$		0.35
	$200 < P \leq 500$		1.0
	$P > 500$		2.0
Tower overhead (psia)	$P > 50$	40–50	0.2–0.5
	Atmospheric	60–100	
	Vacuum ($P < 10$)		0.05–0.1
Compressor	Piping suction	75–200	0.5
	Piping discharge	100–250	1.0
	Gas lines with battery limit		0.5
	Refrigerant suction lines	15–35	
	Refrigerant discharge lines	35–60	
Steam lines	Saturated	200	
	Superheated	250	
	Steam pressure (psig) 0–50	167	0.25
	Steam pressure (psig) 50–150	117	0.40
	Steam pressure (psig) 150–300		1.0
	Steam pressure (psig) > 300		1.5
High-pressure steam lines	Short ($L < 600$ ft)		1.0
	Long ($L > 600$ ft)		0.5
Exhaust steam lines	Exhaust steam lines		0.5
	$P >$ atmosphere		0.5
	Leads to exhaust header		1.5
Relief valve	Discharge	$0.5V_s$	
	Entry point at silencer	V_s	

Source: Coker, A.K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, 1995, p. 164.

TABLE 5.8
***k* Values for Some Common Gases**

Gas	Molecular Weight	$k = C_p/C_v$	Gas	Molecular Weight	$k = C_p/C_v$
Acetylene	26.0	1.30	Hydrogen	2.0	1.41
Air	29.0	1.40	Methane	16.0	1.32
Ammonia	17.0	1.32	Methyl chloride	50.5	1.20
Argon	39.9	1.67	Natural gas	19.5	1.27
Butane	58.1	1.11	Nitric oxide	30.0	1.40
Carbon dioxide	44.0	1.30	Nitrogen	28.0	1.41
Carbon monoxide	28.0	1.40	Nitrous oxide	44.0	1.31
Chlorine	70.9	1.33	Oxygen	32.0	1.40
Ethane	30.0	1.22	Propane	44.1	1.15
Ethylene	28.0	1.22	Propylene	42.1	1.14
Helium	4.0	1.66	Sulfur dioxide	64.1	1.26
Hydrogen chloride	36.5	1.41			

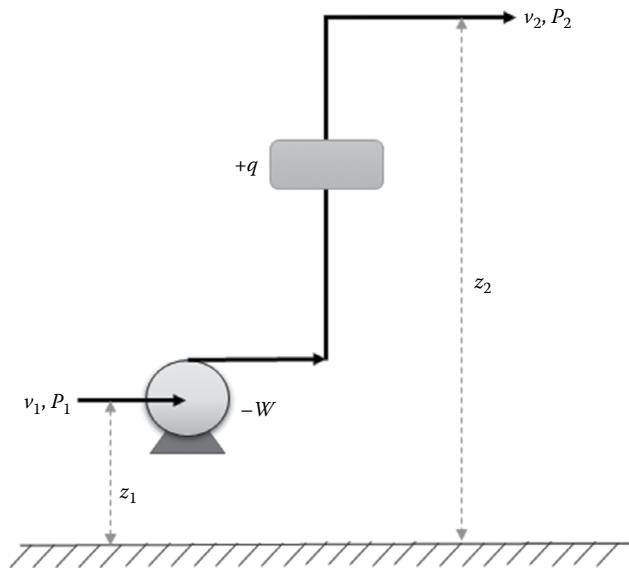


FIGURE 5.20 A single-stream piping system. (From Coker, A.K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, 1995, p. 166.)

$h_L = K(V^2/2g_c)$ and the mass flow rate $G = \rho VA$ are constant. Then the maximum flow rate through the pipe can be expressed as

$$G = 1335.6d^2 \sqrt{\left(\frac{\rho_1}{K_{total} + 2 \ln \frac{P_1}{P_2}} \right) \left(\frac{P_1^2 - P_2^2}{P_1} \right)} \quad (\text{lb/hr})$$

where K_{total} is the total velocity head due to friction, fittings, and valves and is given by

$$K_{total} = f_D \frac{L}{D} + \sum K_f$$

If the pressure drop due to velocity acceleration is relatively small compared with the pressure drop due to friction, the term $\ln P_1/P_2$ can be neglected. Then the pressure drop ΔP can be expressed as

$$\Delta P \cong P_1 - \sqrt{P_1^2 - \frac{P_1 G^2 K_{total}}{\rho_1 (1335.6d^2)^2}} \quad (\text{psi})$$

Table 5.9 shows the friction factors for clean commercial steel pipes with complete turbulent flow.

Example 5.18: Compressible Fluid Flow³⁶

Calculate the maximum flow rate of natural gas through a ruptured exchanger tube.

Data: Exchanger tube = 3/4 in., schedule 160, inside diameter $d = 0.614$ in., tube length = 20 ft, pipe friction factor (complete turbulence) = 0.026, compressibility factor $Z = 0.9$, gas temperature = 100°F, molecular weight = 19.5, pressure in tubes (P_1) = 1110 psig, relief valve set pressure

TABLE 5.9**Friction Factors for Complete Turbulent Flow in Commercial Steel Pipes**

Nominal Size (in.)	Friction Factor (f)	Nominal Size (in.)	Friction Factor (f)
1	0.023	10	0.0136
1.5	0.0205	12	0.0132
2	0.0195	14	0.0125
3	0.0178	16	0.0122
4	0.0165	18	0.12
5	0.016		0.0118
6	0.0152	24	0.0116
8	0.0142		

Source: Coker, A.K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, 1995, p. 167.

on the shell side (P_2) = 400 psig, ratio of specific heat capacities $k = 1.27$, fluid viscosity = 0.012 cP, resistance coefficient due to fittings and valves $K = 2.026$.

Solution

The solution procedure is as follows:

1. Calculate the cross-sectional area A and density ρ .
2. Calculate $K_{total} = f_D \frac{L}{D} + \sum K_f$ where $\sum K_f = K$.
3. Calculate $= 1335.6d^2 \sqrt{\left(\frac{\rho_1}{K_{total} + 2 \ln \frac{P_1}{P_2}} \right) \left(\frac{P_1^2 - P_2^2}{P_1} \right)}$.
4. Calculate $N_{Re} = \frac{6.31G}{\mu D}$, $V = \frac{0.0509G}{d^2 \rho_1}$, $V_s = 223 \sqrt{\frac{kT}{M_w}} = 68.1 \sqrt{\frac{kP_1}{\rho_1}}$.
5. Calculate $= \frac{V}{V_s}$, $R = \frac{15,44S_g}{29}$, $P_c = \frac{G}{11,400d^2} \sqrt{\frac{RT}{k(k+1)}}$.

The script *compflow* performs the calculation procedure.

```
% compflow.m: compressible fluid flow
d = 0.614; Lst = 20; f = 0.026; K = 2.026; Z = 0.9; T = 100; Mw = 19.5;
P1 = 1124.7; P2 = 414.7; k = 1.27; mu = 0.012;
T = T+460; % T(R)
D = d/12; Area = pi*D^2/4; % cross sectional area (ft^2)
rho = P1*Mw/(10.72*Z*T); % density (lb/ft^3)
delP = P1 - P2; Kp = f*Lst/D; Kt = K + Kp; %Ktotal
G = 1335.6*d^2 * sqrt(rho*(P1^2 - P2^2)/(Kt + 2*log(P1/P2))/P1);
Nre = 6.31*G/(D*mu); Vg = 0.0509*G/(rho*d^2);
Vs = 223*sqrt(k*T/Mw); Mach = Vg/Vs; Sg = Mw/29; R = 1544/(29*Sg);
Pc = (G/(11400*d^2))*sqrt(R*T/(k*(k+1)));
G, Vg, Mach, Pc
```

The script *compflow* produces the following outputs:

```
>> compflow
G =
8.3969e+03
```

```

Vg =
279.2890
Mach =
0.2074
Pc =
242.3052

```

We can see that the maximum flow rate is 8396.9 lb/hr and the fluid velocity is 279.289 ft/sec. Since the Mach number at the inlet is 0.2074, the flow is subsonic.

5.7 TWO-PHASE FLOW IN PIPES

5.7.1 FLOW PATTERNS

Flow regimes in horizontal two-phase flow can be classified into seven types: stratified flow, wavy flow, annular flow, plug flow, slug flow, bubble or froth flow, and spray or mist flow. The Baker diagram shown in Figure 5.21 can be referred to determine the type of flow in a process pipeline. Establishing the flow regime involves determining the Baker parameters B_x and B_y from the two-phase system's characteristics and physical properties. These parameters can be expressed as³⁷

$$B_x = 531 \left(\frac{W_L}{W_G} \right) \left(\frac{\sqrt{\rho_L \rho_G}}{\rho_L^{2/3}} \right) \left(\frac{\mu_L^{1/3}}{\sigma_L} \right), \quad B_y = 2.16 \left(\frac{W_G}{A} \right) \frac{1}{\sqrt{\rho_L \rho_G}}$$

where W_L and W_G are the flow rates (lb/hr) in liquid phase and vapor phase, respectively. Table 5.10 shows the characteristic linear velocities of the gas and liquid phases in each flow regime.

The equations representing the boundaries of the flow regimes shown in Figure 5.21 are given below:

$$C_1: \ln B_y = 9.774459 - 0.6548(\ln B_x)$$

$$C_2: \ln B_y = 8.67964 - 0.1901(\ln B_x)$$

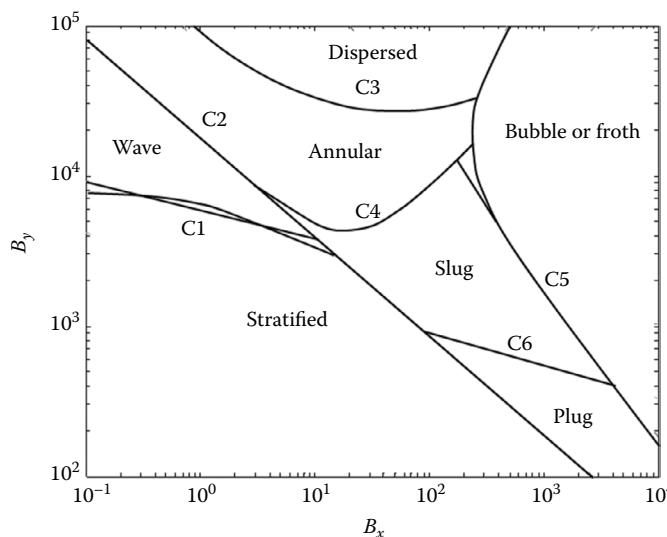


FIGURE 5.21 Baker parameters for two-phase flow regimes. (From Coker, A.K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, 1995, p. 172.)

TABLE 5.10
Characteristic Linear Velocities of Two-Phase Flow Regimes

Flow Regime	Liquid-Phase Velocity (ft/sec)	Vapor-Phase Velocity (ft/sec)
Bubble or froth	5–15	0.5–2
Plug	2	<4
Stratified	<0.5	0.5–10
Wave	<1.0	>15
Slug	15	3–50
Annular	<0.5	>20
Dispersed, spray, or mist	Close to vapor velocity	>200

Source: Coker, A.K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, 1995, p. 174.

$$C_3: \ln B_y = 11.3976 - 0.6084(\ln B_x) + 0.0779(\ln B_x)^2$$

$$C_4: \ln B_y = 10.7448 - 1.6265(\ln B_x) + 0.2839(\ln B_x)^2$$

$$C_5: \ln B_y = 14.569802 - 1.0173(\ln B_x)$$

$$C_6: \ln B_y = 7.8206 - 0.2189(\ln B_x)$$

The Baker parameter B_y for two-phase flow regimes are calculated by the MATLAB function *twophreg*. This function computes B_y for a given B_x , which can be a vector.

```
function By = twophreg(Bx)
% 2-phase flow regimes
% Bx, By: Baker parameters

By = [];
C1 = exp(9.774459 - 0.6548*log(Bx));
C2 = exp(8.67964 - 0.1901*log(Bx));
C3 = exp(11.3976 - 0.6084*log(Bx) + 0.0779*log(Bx).^2);
C4 = exp(10.7448 - 1.6265*log(Bx) + 0.2839*log(Bx).^2);
C5 = exp(14.569802 - 1.0173*log(Bx));
C6 = exp(7.8206 - 0.2189*log(Bx));
By = [C1' C2' C3' C4' C5' C6'];
```

5.7.2 PRESSURE DROP

The two-phase pressure drop in horizontal pipes, ΔP_T , can be expressed as

$$\frac{\Delta P_T}{100 \text{ ft}} = \frac{\Delta P_G}{100 \text{ ft}} \cdot Y_G \quad (\text{psi}/100 \text{ ft})$$

where

$\Delta P_G/100 \text{ ft}$ (psi/100 ft) is the pressure drop of gas if flowing alone in the pipe
 Y_G is the two-phase flow modulus

Y_G is a function of the Lockhart–Martinelli two-phase flow modulus X , which is defined as

$$X = \sqrt{\frac{\Delta P_L}{\Delta P_G}}, \quad Y_G = f(X)$$

where $\Delta P_L/100$ ft (psi) is the pressure drop of liquid if flowing alone in the pipe. The equations of Y_G for different flow regimes are shown in [Table 5.11](#).

For wave flow, the Huntington friction factor F_H is used to determine the two-phase pressure loss:

$$H_x = \left(\frac{W_L}{W_G} \right) \left(\frac{\mu_L}{\mu_G} \right), \quad \ln F_H = 0.2111 \ln(H_x) - 0.3993$$

$$\frac{\Delta P_T}{100 \text{ ft}} = \frac{0.000366 F_H W_G^2}{d^5 \rho_G}$$

where

W_L and W_G are the flow rates of liquid and vapor phases (lb/hr), respectively

F_H is the Huntington friction factor

A is the inside cross-sectional area of the pipe (ft²)

ρ_G is the density of vapor (lb/ft²)

μ_L and μ_G are the viscosities of liquid and vapor phases (cP), respectively

TABLE 5.11
Equations of Y_G for Two-Phase Flow Regimes

Flow Regime	Y_G
Bubble/froth	$Y_G = \left[\frac{14.2X^{0.75}}{(W_L/A)^{0.1}} \right]^2$
Plug	$Y_G = \left[\frac{27.315X^{0.855}}{(W_L/A)^{0.17}} \right]^2$
Stratified	$Y_G = \left[\frac{15,400X}{(W_L/A)^{0.8}} \right]^2$
Slug	$Y_G = \left[\frac{1190X^{0.815}}{\sqrt{W_L/A}} \right]^2$
Annular	$Y_G = (aX^b)^2, a = 4.8 - 0.3125d, b = 0.343 - 0.021d$ $d = \text{pipe inside diameter (in.) (if } d > 12, \text{ set } d = 10)$
Dispersed/spray	$Y_G = [\exp(C_0 + C_1(\ln X) + C_2(\ln X)^2 + C_3(\ln X)^3)]^2$ $C_0 = 1.4659, C_1 = 0.49138, C_2 = 0.04887, C_3 = -0.000349$

Source: Coker, A.K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, 1995, pp. 177–178.

The pressure drop of liquid or vapor flowing alone in a straight pipe can be expressed as

$$\frac{\Delta P_T}{100 \text{ ft}} = \frac{0.000366 f_D W_x^2}{d^5 \rho_G} \quad (\text{psi}/100 \text{ ft})$$

where W_x is the flow rate of liquid or vapor (lb/hr). Modification of this relation yields the relationship representing the overall pressure drop of the two-phase flow for the total length of pipe plus fittings based on the gas-phase pressure drop:

$$\Delta P_{overall} = \frac{0.000366 \cdot f_D \cdot W_G^2 \cdot Y_G \cdot L}{100 d^5 \rho_G} \quad (\text{psi})$$

The velocity of the two-phase fluid can be given by

$$V = \frac{0.0509}{d^2} \left(\frac{W_G}{\rho_G} + \frac{W_L}{\rho_L} \right)$$

5.7.3 CORROSION AND EROSION

High velocities in a two-phase flow system lead to corrosion and erosion. An index based on velocity head can indicate whether corrosion or erosion may become significant at a particular velocity.⁴¹ This index can be used to determine the range of mixture densities and velocities below which corrosion–erosion should not occur. This index is

$$\rho_m U_M^2 \leq 10,000$$

The mixture density ρ_m and the mixture velocity U_M are given by

$$\rho_m = \frac{W_L + W_G}{\left(\frac{W_L}{\rho_L} + \frac{W_G}{\rho_G} \right)}, \quad U_M = U_G + U_L = \frac{1}{3600 A} \left(\frac{W_G}{\rho_G} + \frac{W_L}{\rho_L} \right)$$

where U_L and U_G are the velocities of liquid and vapor (ft/sec), respectively.

Example 5.19: Pressure Drop in Two-Phase Flow⁴²

Determine the pressure drop of a 5 mile (26,400 ft) length in a 6 in. schedule 40 ($ID = 6.065$ in.) pipe, for a 5,000 bpd (77,956 lb/hr) rate of salt water ($\gamma_w = 1.07$), having a vapor flowing at 6 mmsecf (12,434 lb/hr) of 0.65 specific gravity at a temperature of 110°F.

Physical Properties	Liquid	Vapor
Flow rate (lb/hr)	77,956	12,434
Density (lb/ft ³)	66.7	2.98
Viscosity (cP)	1.0	0.02
Surface tension (dyne/cm)	70.0	

Solution

The script *twophdP* calculates the pressure drop. This script calls the function *twophmod* to determine two-phase flow regime and two-phase flow modulus Y_G . The function *twophmod* includes subfunctions to calculate the value of Y_G in each flow regime.

```
twophdP.m
% twophdP.m: pressure drop in two-phase flow
% data
clear all;
rholf = 66.7; rhogf = 2.98; mul = 1; mug = 0.02; siglf = 70;
L = 26400; d = 6.065; Wlf = 77956; Wgf = 12434;
D = d/12; eD = 0.00015/D;

% liquid/vapor phase friction factor
Nrelf = 6.31*Wlf/(d*mul); Nregf = 6.31*Wgf/(d*mug);
if Nrelf <= 2100
    fLf = 64/Nrelf;
else
    Avf = eD/3.7 + (6.7/Nrelf)^0.9;
    fLf = 4./(-4*log10(eD/3.7 - 5.02*log10(Avf)/Nrelf)).^2;
end
if Nregf <= 2100
    fGf = 64/Nregf;
else
    Avf = eD/3.7 + (6.7/Nregf)^0.9;
    fGf = 4./(-4*log10(eD/3.7 - 5.02*log10(Avf)/Nregf)).^2;
end%
Determine two-phase flow modulus (Yg) and flow regimes
fr = {'stratified', 'wave', 'plug', 'slug', 'bubble', 'annular', 'dispersed'};
rind = 7;

delPlf = 3.66e-4 * fLf * Wlf^2/(d^5 * rholf);
delPgf = 3.66e-4 * fGf * Wgf^2/(d^5 * rhogf);
[Ygf rindf] = twophmod(rholf, rhogf, mul, siglf, Wlf, Wgf, delPlf, delPgf, d);
if rindf == 2
    fHf = exp(0.2111*log(Wlf*mul/(Wgf*mug)) - 3.993);
    delPt = 3.66e-4 * fHf * Wgf^2 * L/(d^5 * rhogf * 100);
else
    delPt = Ygf*delPgf*L/100;
end
fprintf('Flow regime: %s\n', fr{rindf});
fprintf('Two-phase flow modulus (Yg): %g\n', Ygf);
fprintf('Total pressure drop(psi): %g\n', delPt);

twophmod.m
function [Yg rind] = twophmod(rhol, rhog, mul, sigl, Wl, Wg, delPl, delPg, d)
% Calculation of two-phase flow modulus (Yg)
% input:
% rhol, rhog: density of liquid and vapor (lb/ft^3)
% mul: liquid viscosity (cP)
% sigl: liquid surface tension (dyne/cm)
% Wl, Wg: flow rates of liquid and vapor (lb/h)
% delPl, delPg: pressure drop for liquid-only/vapor-only flow
% d: pipe inside diameter (in)
% output:
% Yg: two-phase flow modulus
% rind: flow regime index

% Calculate Baker parameter Bx and By
D = d/12; A = pi*D^2/4;
```

```

Bx = 531*(Wl/Wg)*(sqrt(rhol*rhog)/(rhol^(2/3)))*(mul^(1/3)/sigl);
By = 2.16*(Wg/A)/sqrt(rhol*rhog);
C = twophreg(Bx);

% Classify flow regimes and calculate Yg for each flow regime
x = sqrt(delPl/delPg); Wa = Wl/A;
if By <= C(1)
    if By <= C(2) % By < C1,C2
        [Yg rind] = strat(x,Wa);
    else % C2 < By < C1
        [Yg rind] = wave(x,Wa);
    end
else % C1 < By
    if By < C(5) % C1 < By < C5
        if By < C(6) % C1 < By < C5,C6
            [Yg rind] = plug(x,Wa);
        else % C1,C6 < By < C5
            if By < C(4) % C1,C6 < By < C4,C5
                [Yg rind] = slug(x,Wa);
            else % C1,C4,C6 < By < C5
                if By <= C(3) % C1,C4,C6 < By < C3,C5
                    [Yg rind] = annul(x,d);
                else % C1,C3,C4,C6 < By < C5
                    [Yg rind] = dispr(x);
                end
            end
        end
    end
else % C1,C5 < By
    if Bx > 150
        [Yg rind] = bubb(x,Wa);
    else % Bx <= 150
        if By <= C(3) % C1,C5 < By < C3
            [Yg rind] = annul(x,d);
        else % C1,C3,C5 < By
            [Yg rind] = dispr(x);
        end
    end
end
end

function [Yg rind] = strat(x,Wa)
rind = 1; Yg = (15400*x/(Wa^0.8))^2;
end

function [Yg rind] = wave(x,Wa)
rind = 2; Yg = 0;
end

function [Yg rind] = plug(x,Wa)
rind = 3; Yg = (27.315*x^0.855 / Wa^0.17)^2;
end

function [Yg rind] = slug(x,Wa)
rind = 4; Yg = (1190*x^0.815/sqrt(Wa))^2;
end

function [Yg rind] = bubb(x,Wa)
rind = 5; Yg = (14.2*x^0.75 / Wa^0.1)^2;
end

function [Yg rind] = annul(x,d)
rind = 6; dx = d; if d > 12, dx = 10; end;

```

```

Yg = ((4.8 - 0.3125*dx)*x^(0.343-0.021*dx))^2;
End

function [Yg rind] = dispr(x)
rind = 7; a0 = 1.4659; a1 = 0.49138; a2 = 0.04887; a3 = -0.000349;
Yg = (exp(a0 + a1*log(x) + a2*(log(x))^2 + a3*(log(x))^3))^2;
end

```

The script *twophdP* produces the following outputs:

```

>> twophdP
Flow regime: annular
Two-phase flow modulus (Yg): 10.0125
Total pressure drop(psi): 97.7391

```

5.7.4 VAPOR–LIQUID TWO-PHASE VERTICAL DOWNFLOW

In a vertical downflow, large vapor bubbles, known as slug flow, are formed in the liquid stream. With bubbles greater than 1 in. in diameter and the liquid viscosity less than 100 cP, slug flow can be represented by the Froude numbers $N_{Fr,L}$ (for liquid) and $N_{Fr,G}$ (for vapor). These numbers are defined as

$$N_{Fr,L} = \frac{V_L}{\sqrt{gD}} \sqrt{\frac{\rho_L}{\rho_L - \rho_G}}, \quad N_{Fr,G} = \frac{V_G}{\sqrt{gD}} \sqrt{\frac{\rho_L}{\rho_L - \rho_G}}$$

where D is the inside diameter of the pipe (ft). V_L and V_G are the superficial velocities based on the total pipe cross section and are given by

$$V_L = \frac{W_L}{3600\rho_L A}, \quad V_G = \frac{W_G}{3600\rho_G A} \quad (\text{ft/sec})$$

where $A = \pi D^2/4$ (ft²). If $N_{Fr,L} < 0.31$, the vertical pipe is self-venting and the bubbles rise. If the Froude number is in the range of $0.3 < N_{Fr,L} < 1.0$, the flow is pulse flow and pressure pulsation and vibration are produced. If $N_{Fr,L} > 1.0$, the friction force offsets the effect of gravity and thus requires no pressure gradient in the vertical downflow liquid.⁴³

5.7.5 PRESSURE DROP IN FLASHING STEAM CONDENSATE FLOW

When a liquid is flowing near its saturation point in a pipeline, decreased pressure will cause vaporization. The higher the pressure difference, the greater the vaporization resulting in flashing of the liquid. The pressure drop for flashed condensate mixture can be determined by employing the following calculation procedures:

1. Calculate the flashed steam rate W_G (lb/hr) and flashed condensate rate W_L (lb/hr):

$$W_{FL} = B \left(\ln P_C \right)^2 - A$$

$$A = 0.00671 \left(\ln P_h \right)^{2.27}, \quad B = 0.0088 + 10^{-4} e^X, \quad X = 6.122 - \left(\frac{16.919}{\ln P_h} \right)$$

$$W_G = W_{FL} \cdot W, \quad W_L = W - W_G$$

where

- W_{FL} is the weight fraction of condensate flashed to vapor
- P_C is the steam condensate pressure before flashing (psia)
- P_h is the flashed condensate header pressure (psia)

The temperature of flashed condensate, T_{FL} (°F), is given by

$$T_{FL} = 115.68(P_h)^{0.226}$$

2. Determine the density of flashed vapor ρ_G (lb/ft³) and flashed condensate ρ_L (lb/ft³) and the density of flashed condensate/vapor mixture ρ_M (lb/ft³):

$$\rho_G = 0.0029P_h^{0.938}, \quad \rho_L = 60.827 - 0.078P_h + 0.00048P_h^2 - 0.0000013P_h^3$$

$$\rho_M = \frac{W_G + W_L}{\left(\frac{W_G}{\rho_G} + \frac{W_L}{\rho_L} \right)}$$

3. Calculate the pressure drop ΔP_T (psi/100 ft). For turbulent flow,

$$f = -\frac{0.25}{\left[-\log\left(\frac{0.000486}{d} \right) \right]^2} \quad (d: \text{internal pipe diameter (in.)})$$

$$V = \frac{3.054}{d^2} \left[\frac{W_G}{\rho_g} + \frac{W_L}{\rho_L} \right], \quad \Delta P_T = \frac{0.000336fW^2}{d^5\rho_m}$$

where V is the velocity of flashed condensate mixture (ft/min). If $V \geq 5000$ ft/min, the condensate mixture may cause some problems to the piping system.

5.8 FLOW THROUGH PACKED BEDS

The pressure drop, ΔP_T (lb/in.²), in the flow of fluids through packed beds of granular particles can be obtained from the Ergun equation⁴⁴:

$$\Delta P_T = \frac{B_L}{144} \cdot \frac{1-\epsilon}{\epsilon^3} \cdot \frac{G^2}{D_p g_c \rho} \left[\frac{362.85\mu(1-\epsilon)}{D_p G} + 1.75 \right]$$

$$\epsilon = \frac{\rho_c - \rho_b}{\rho_c}, \quad \rho = \frac{M_w P}{10.73 Z T}, \quad D_p = \frac{6(1-\epsilon)}{S}, \quad S = \frac{1-\epsilon}{V_p} \cdot A_p$$

$$V_p = \frac{1}{1728} \left(P_D^2 \cdot \frac{\pi}{4} \cdot P_L \right), \quad A_p = \frac{1}{144} \left(\frac{\pi P_D^2}{2} + \pi P_D P_L \right), \quad N_{Re} = \frac{D_p G}{2.419 \mu (1-\epsilon)}$$

where

B_L is the length of the packed bed (ft)

ϵ is the void fraction of the packed bed

ρ_c is the catalyst density (lb/ft³)

ρ_b is the density of the packed bed (lb/ft³)

ρ is the fluid density (lb/ft³)

D_p is the effective particle diameter (ft)

S is the surface area of the packed bed (ft²/ft³ bed)

M_w is the molecular weight of the fluid

P is the fluid pressure (psia)

T is the fluid temperature (°R)

A_p is the particle area (ft²)

V_p is the particle volume (ft³)

P_d is the particle diameter (in.)

P_L is the particle length (in.)

Z is the compressibility factor

G is the superficial fluid mass velocity (lb/(hr ft²) and is given by $G = \frac{W}{A}$

W is the mass flow rate of the fluid (lb/hr)

A is the cross-sectional area of the packed bed (ft²)

Example 5.20: Pressure Drop for Flow through a Packed Bed⁴⁵

Estimate the pressure drop in a 60 ft length of 1½ (schedule 40, inside diameter = 1.61 in. (=0.134 ft)) pipe packed with catalyst pellets 1/4 in diameter when 104.4 lb/hr of gas is passing through the bed. The temperature is constant along the length of the pipe at 260°C. The void fraction is 45% and the properties of the gas are similar to those of air at this temperature (density = 0.413 lb/ft³, viscosity = 0.0278 cP). The entering pressure is 10 atm.

Solution

The MATLAB function *dpcatbed* calculates the pressure drop.

```
function dP = dpcatbed(W,Bd,B1,Pd,P1,ep,mu,rho)
% Calculates the pressure drop for flow through a packed bed

gc = 4.17e8; A = pi*Bd^2/4;
G = W/A; % superficial mass flow rate (lb/h/ft^2)
Ap = (pi*Pd^2/2 + pi*Pd*P1)/144; Vp = pi*P1*Pd^2/(4*1728);
S = Ap*(1-ep)/Vp; ePd = 6*(1-ep)/S;
Nre = G*ePd/(2.419*mu*(1 - ep));
if Nre <1
    fP = 150/Nre;
elseif Nre < 1e4
    fP = 150/Nre + 1.75;
else
    fP = 1.75;
end
dP = B1*(1-ep)*G^2*(150*2.419*mu*(1-ep)/ePd/G + 1.75) / ...
(144*ep^3*ePd*gc*rho);
end
```

The following commands produce desired outputs:

```
>> Bd = 0.134; B1 = 60; Pd = 0.25; P1 = 0.25; ep = 0.45;
>> W = 104.4; mu = 0.0278; rho = 0.413;
```

```

>> dPt = dpcatbed(W,Bd,B1,Pd,Pl,ep,mu,rho)
dPt =
68.6034

```

We can see that the total pressure drop is 68.6034 lb/in.²

PROBLEMS

- 5.1** Water flows inside a horizontal circular pipe at 25°C. Determine the average velocity and plot the velocity distribution inside the pipe. The length and the inside radius of the pipe are 12 and 0.01 m, respectively, the viscosity of water is 8.937×10^{-4} kg/(m · sec), and the pressure drop is $\Delta P = 520$ Pa.
- 5.2** A non-Newtonian fluid is flowing in a horizontal pipe where $L = 10$ m, $r = 0.009$ m, n (flow behavior index) = 0.46, $k = 0.012$ N · secⁿ/m², and $\Delta P = 100$ Pa. Calculate the average velocity and plot the velocity profile versus radius.
- 5.3** Water flows down an inclined flat plate of depth $L = 0.004$ m and unit width, as shown in Figure P5.3. For water, the density is 1000 kg/m³ and the viscosity is 9.93×10^{-4} Pa · sec. Plot v_x versus z when $\theta = 30^\circ$.
- 5.4** Water flows inside a smooth horizontal circular pipe at 21°C. The inside diameter of the pipe is 0.1 m and the velocity is 0.5 m/sec. Calculate the pressure drop when the pipe length is 200 m. At 21°C, the density of water is $\rho = 997.92$ kg/m³ and the viscosity is $\mu = 0.000982$ kg/m/sec. Assume that the friction factor is given by $f = 0.0468N_{Re}^{-0.2}$.
- 5.5** Determine the friction factors using various correlation equations (i.e., SC, CB, CBW, HA, CH, NK, and BS equation) at $N_{Re} = 2 \times 10^4$ and $N_{Re} = 3.2 \times 10^7$ for pipes where $\epsilon/D = 0.0001$.
- 5.6** Water flows inside a smooth pipeline at 25°C with a flow rate of 3 liter/sec. The length of the pipeline is 120 m and the value of the maximum possible pressure drop is $\Delta P = 100$ kPa. The density and the viscosity of water at 25°C are 994.6 kg/m³ and $\mu = 8.931 \times 10^{-4}$ kg/(m · sec), respectively. Determine the inside diameter of the pipe. The Fanning friction factor is given by $f = 16/N_{Re}$ for laminar flow and by $1/\sqrt{f} = 4.0 \log(N_{Re}\sqrt{f}) - 0.4$ (Nikuradse equation) for turbulent flow. Determine the diameter of the pipeline.
- 5.7** Figure P5.7 shows a pipeline that delivers water at constant temperature T from point 1, where the pressure is p_1 and the elevation is z_1 , to point 2, where the pressure is p_2 and the elevation is z_2 .⁴⁶ The density ρ (lb_m/ft³) and viscosity μ (lb_m/(ft · sec)) of water at T are given by

$$\rho = 62.122 + 0.0122T - 1.54 \times 10^{-4}T^2 + 2.65 \times 10^{-7}T^3 - 2.24 \times 10^{-10}T^4$$

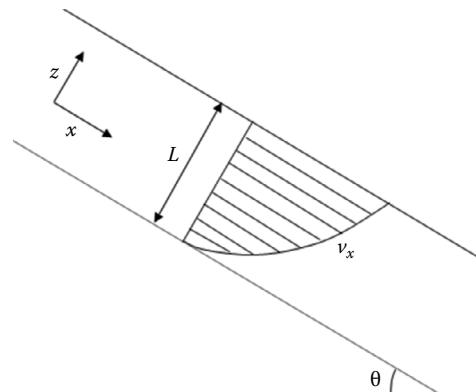


FIGURE P5.3 Flow down an inclined plate.

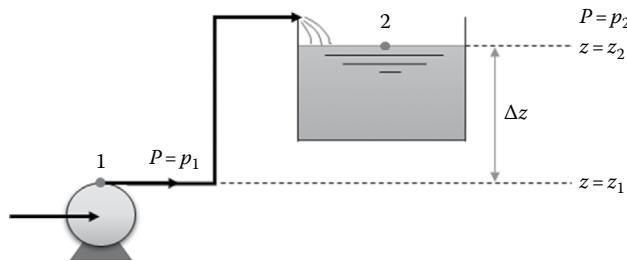


FIGURE P5.7 Water flow in a pipeline. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 110.)

$$\ln \mu = -11.0318 + \frac{1057.51}{T + 214.624}$$

At point 1, $p_1 = 150$ psig and $z_1 = 0$ ft, and at point 2, $p_2 = 0$ psig and $z_2 = 300$ ft. Assume that $T = 60^\circ\text{F}$.

1. Calculate the flow rate q (gal/min) of water when the water is delivered through 8 in. schedule 40 pipe with $\epsilon = 0.00015$ ft. The effective length of the pipe is $L = 5000$ ft.
 2. Plot the calculated flow rates as a function of pressure difference $\Delta p = p_2 - p_1$ when p_1 increases up to 200 psig ($p_1 \leq 200$ psig). What is the minimal value of p_1 needed to start flow?
- 5.8** [Figure P5.8](#) shows a pipeline isometric of a 6 in. schedule 40 steel pipeline with six 90° LR (long radius) elbows and two flow-through tees. The actual length of the pipe is 78 ft. Kerosene flows through the pipeline with a flow rate of 1026 gpm at 321°F. The density of kerosene at 60°F is 51 lb/ft³, and the specific gravity of kerosene at 60°F and at 321°F are 0.82 and 0.72, respectively. The viscosity of kerosene at 321°F is 0.3 cP. Determine the pressure drop between point A and point B.
- 5.9** A pipeline network with two flow loops is illustrated in [Figure P5.9](#). The lengths and internal diameters of the pipes are shown in [Table P5.9](#). Calculate the mass flow rate in each flow loop.

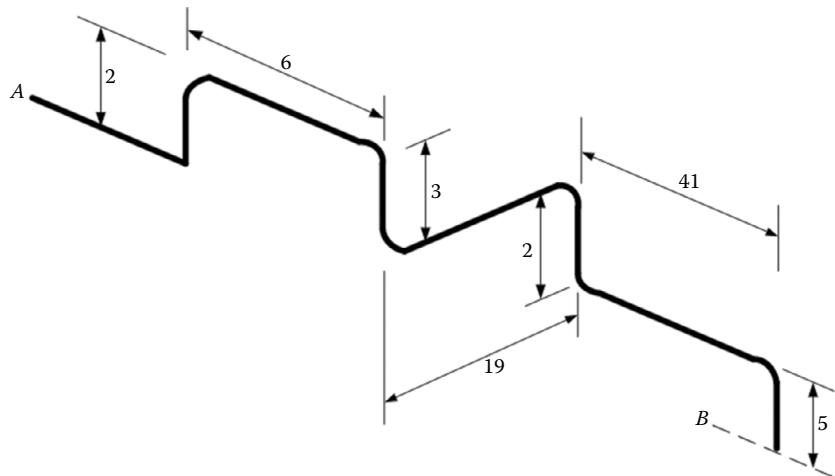


FIGURE P5.8 Kerosene flow in a pipeline. (From Coker, A.K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, 1995, p. 195.)

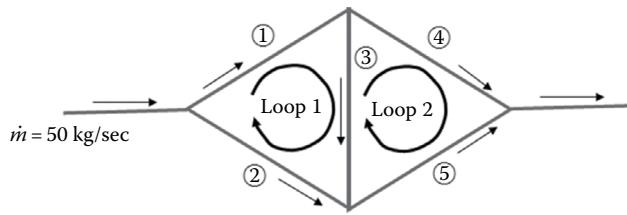


FIGURE P5.9 Two-loop network. (From Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, 1985, p. 230.)

TABLE P5.9
Lengths and Internal Diameters of the Pipes

Flow (<i>i</i>)	Diameter (D_i , m)	Length (L_i , m)
1	0.051	17
2	0.061	17
3	0.04	20
4	0.061	19
5	0.051	19

A pipe roughness of $\epsilon = 5 \times 10^{-6}$ m is assumed uniformly for all pipes. The density and viscosity of the fluid are 991 km/m^3 and $1.2 \times 10^{-4} \text{ N} \cdot \text{sec/m}^2$, respectively, and the inlet flow rate to the network is $\dot{m}_0 = 0.5 \text{ kg/sec}$.

- 5.10** Calculate the Froude number and the flow condition for the two-phase flow in a 6 in. (schedule 40) vertical pipe. The flow rates of liquid and vapor are $W_L = 6930 \text{ lb/hr}$ and $W_G = 1444 \text{ lb/hr}$, respectively, and the densities of liquid and vapor are $\rho_L = 61.8 \text{ lb/ft}^3$ and $\rho_G = 0.135 \text{ lb/ft}^3$, respectively.
- 5.11** In an open tank flow system, $\rho = 1000 \text{ kg/m}^3$, $A = 0.465 \text{ m}^2$, $z_0 = 3.05 \text{ m}$, $g = 9.8 \text{ m/sec}^2$, $C_{v1} = C_{v2} = 0.012 \text{ m}^2/\text{N}^{1/2}/\text{sec}$, $P_{10} = P_1 = 1.38 \times 10^5 \text{ N/m}^2$, $P_{30} = P_3 = 1.08 \times 10^5 \text{ N/m}^2$, $P_0 = 1.014 \times 10^5 \text{ N/m}^2$. Plot the dimensionless level z^* , the dimensionless flow rates F_1^* and F_2^* , and the dimensionless pressure P_2^* versus dimensionless time t^* .⁴⁷
- 5.12** Calculate the pressure drop for each 6 in. (schedule 40) condensate header when the flow rate = 10,000 lb/hr, steam condensate pressure (P_c) = 114.7 psia, and the header pressure (P_h) = 14.7 psia.

REFERENCES

1. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice-Hall, Boston, MA, p. 85, 2003.
2. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 294, 2008.
3. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 295, 2008.
4. Bird, R. B., W. E. Stewart, and E. N. Lighfoot, *Transport Phenomena*, 2nd ed., John Wiley & Sons, Hoboken, NJ, p. 44, 2002.
5. de Nevers, N., *Fluid Mechanics for Chemical Engineers*, 3rd ed., McGraw-Hill, New York, NY, p. 211, 2005.
6. Shacham, M., A review of non-iterative friction factor correlations for the calculation of pressure drop in pipes, *Industrial & Engineering Chemistry Fundamentals*, 19, 228–229, 1980.
7. Colebrook, C. F., Turbulent flow in pipes with particular references to the transition region between the smooth and rough pipe laws, *Journal of the Institution of Civil Engineers (London)*, 11, 133, 1938–1939.

8. Colebrook, C. F. and C. M. White, Characteristics of flow in the transition region between the smooth and rough pipe, *Journal of the Institution of Civil Engineers (London)*, 10, 99–108, 1938–1939.
9. Haaland, S. E., Simple and explicit formulas for the friction factor in turbulent flow, *Transactions of the ASME, Journal of Fluids Engineering*, 105, 89, 1983.
10. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 158, 1995.
11. de Nevers, N., *Fluid Mechanics for Chemical Engineers*, 3rd ed., McGraw-Hill, New York, NY, p. 187, 2005.
12. Churchill, S. W., Friction factor equations spans all fluid-flow regimes, *Chemical Engineering*, 84(24), 91, 1977.
13. Nikuradse, J., New development in pipe flow optimization modeling, *VDI-Forschungsheft*, 65(4), 356, 1932.
14. Kapuno, R. R. A., *Programming for Chemical Engineers*, Infinity Science Press, Hingham, MA, p. 105, 2008.
15. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 110, 2008.
16. Hooper, W. B., The two-K method predicts head loss in pipe fittings, *Chemical Engineering*, August 24, 96–100, 1981.
17. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 155, 1995.
18. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 157, 1995.
19. Bird, R. B., W. E. Stewart, and E. N. Lightfoot, *Transport Phenomena*, 2nd ed., John Wiley & Sons, Inc., Hoboken, NJ, p. 207, 2002.
20. Bird, R. B., W. E. Stewart and E. N. Lightfoot, *Transport Phenomena*, 2nd ed., John Wiley & Sons, Inc., Hoboken, NJ, pp. 207–208, 2002.
21. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, pp. 194–195, 1995.
22. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 309, 2008.
23. Lang, F. D. and B. L. Miller, Pipe network analysis based on the generalized friction factor correlation of Churchill, *Chemical Engineering*, 88(13), 95, 1981.
24. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, pp. 231–232, 1985.
25. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, pp. 234–235, 1985.
26. Ramirez, W. F., *Computational Methods for Process Simulation*, 2nd ed., Butterworth Heinemann, Oxford, UK, pp. 155–157, 1997.
27. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, p. 226, 1985.
28. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, p. 225, 1985.
29. Tomita, Y., A study of non-Newtonian flow in pipe lines, *Bulletin of the Japan Society of Mechanical Engineers*, 2, 10, 1959.
30. Dodge, D. W. and A. B. Metzner, Flow non-Newtonian fluids correlation of laminar, transition and turbulent flow regions, *AIChE Journal*, 5, 189, 1959.
31. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, p. 227, 1985.
32. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, pp. 229–230, 1985.
33. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 163, 1995.
34. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 164, 1995.
35. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 167, 1995.
36. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 198, 1995.
37. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 175, 1995.

38. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 172, 1995.
39. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 174, 1995.
40. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, pp. 177–178, 1995.
41. Coulson, J. M. and J. F. Richardson, *Chemical Engineering*, Vol. 1, 3rd ed., Pergamon Press, Oxford, UK, pp. 91–92, 1978.
42. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, pp. 200–201, 1995.
43. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 182, 1995.
44. Ergun, S., Fluid flow through packed columns, *Chemical Engineering Progress*, 48(2), 89–92, 1952.
45. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 175–176, 2011.
46. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 307, 2008.
47. Ramirez, W. F., *Computational Methods for Process Simulation*, 2nd ed., Butterworth Heinemann, pp. 150–151, Oxford, UK, 1997.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

6 Chemical Reaction Engineering

Chemical reactions are essential technological elements in a wide range of industries. Chemical reactions are indispensable in converting less valuable raw materials into higher-value products and in converting one form of energy to another. Many reactions of industrial significance are complex, involving side reactions—which often result in the formation of undesirable by-products. Chemical kinetics allows us to understand how reaction rates depend on variables such as concentration, temperature, and pressure. Most industrial chemical processes involve some kind of catalyst in order to increase the reaction rates. The behavior of heterogeneous catalysts can be significantly influenced by the rate of heat and mass transfer to and from the sites in the catalyst where the reaction occurs.

The design and analysis of chemical reactors is built upon a sound understanding of chemical kinetics. In the operation of existing reactors, periodic analysis of the reactor performance is necessary to ensure optimal operation under varying operating conditions. It is difficult to generalize chemical reaction kinetics because almost every reaction is unique, characterized by a specific chemical model, involving one or more phases, and invariably taking place in one of a great variety of reactor configurations. Problems in chemical kinetics and the design and analysis of chemical reactors are nonlinear and therefore require the solution of complicated algebraic and differential equations by numerical methods, which can be performed well by MATLAB® programs.

This chapter demonstrates some basic methods that are used for calculating many different classes of chemical reactions typical in industrial situations. The main objective of this chapter is to provide readers with various chemical reaction engineering models and algorithms as well as corresponding MATLAB programs, which have found or can potentially be used for academic or industrial applications. Throughout this chapter, differential and integral methods are used primarily in analyzing reactor data. The MATLAB programs listed here can be used in undergraduate or graduate courses on chemical reaction engineering. Researchers and practicing engineers in the field of chemical engineering can apply these MATLAB programs in the analysis and design of chemical reactors.

6.1 REACTION RATES

6.1.1 REACTION RATE CONSTANT

The Arrhenius equation for the reaction rate constant is given by

$$k_A = Ae^{-E/(RT)} \quad \text{or} \quad \ln k_A = \ln A - \frac{E}{R} \left(\frac{1}{T} \right)$$

where

A is the frequency factor

E is the activation energy (J/mol)

R is gas constant ($=8.314 \text{ J/(mol} \cdot \text{K)}$)

T is absolute temperature (K)

A and E can be determined by using experimental data for reaction rates at different temperatures.

Example 6.1: Estimation of the Activation Energy¹

Estimate the activation energy for the decomposition of benzene diazonium chloride to produce chlorobenzene and nitrogen using the information given in [Table 6.1](#) for this 1st-order reaction.

Solution

We start by recalling the relation

$$\ln k = \ln A - \frac{E}{R} \left(\frac{1}{T} \right)$$

The plot of $\ln k$ versus $1/T$ yields a straight line. Thus, A and E can be obtained from the equation of the straight line. Parameters of a 1st-order line can be calculated by the built-in function *polyfit*, which performs regression analysis. The script *rateconst* generates a plot of experimental data versus temperature and the resulting straight line.

```
% rateconst.m
k = 1e-3*[0.43 1.03 1.80 3.55 7.17]; y = log(k);
T = [313 319 323 328 333]; x = 1./T;
c = polyfit(x,y,1); A = exp(c(2)), E = -c(1)*8.314
xv = linspace(min(x),max(x),100); yv = polyval(c,xv);
plot(xv,yv,x,y, 'o'), xlabel('1/T(K^-1)'), ylabel('ln(k)')
```

Execution of the script *rateconst* gives A and E and yields the plot of the line and the data points shown in [Figure 6.1](#):

```
>> rateconst
A =
8.0303e+16
E =
1.2148e+05
```

We can see that $A = 8.0303 \times 10^{16} \text{ sec}^{-1}$ and $E = 1.2148 \times 10^5 \text{ J/mol} = 121.48 \text{ kJ/mol}$.

6.1.1.1 Estimation of Reaction Order and Rate Constant Using Experimental Data

Consider a reaction whose rate is given by

$$\frac{dC_A}{dt} = -k' C_A^\alpha$$

Integration of this equation yields

$$t = \frac{1}{k'(1-\alpha)} \left[C_{A0}^{1-\alpha} - C_A^{1-\alpha} \right]$$

TABLE 6.1
Reaction Rate Constant for the Decomposition Reaction
of Benzene Diazonium Chloride

T (K)	313.0	319.0	323.0	328.0	333.0
k (sec $^{-1}$)	0.00043	0.00103	0.00180	0.00355	0.00717

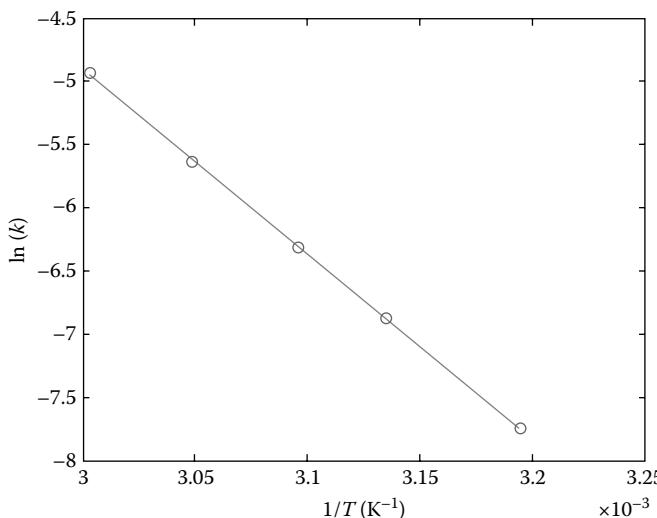


FIGURE 6.1 Reaction rate constant versus temperature.

Using concentration (C_A)–time data obtained in batch experiments, k' and α can be calculated from the solution of the corresponding nonlinear equation system.

Example 6.2: Estimation of Rate Constant and Reaction Order²

Concentration (C_A)–time data were obtained in batch experiments for the liquid-phase reaction $A + B \rightarrow C$.

t (min)	0	50	100	150	200	250	300
C_A (mol/dm ³)	0.05	0.038	0.0306	0.0256	0.0222	0.0195	0.0174

For this reaction, the rate can be represented as $-r_A = k' C_A^\alpha$. The concentration of B is assumed to be constant. Estimate the reaction order α and the rate constant k' .

Solution

At the point (t_i, C_{Ai}) , the nonlinear equation

$$f(k', \alpha) = t_i k' (1 - \alpha) - \left[C_{A0}^{1-\alpha} - C_{Ai}^{1-\alpha} \right] = 0$$

is to be solved. If the number of data points is n , then we have to solve the nonlinear equation system consisting of n equations. The nonlinear equation system is defined by the MATLAB® function *regorder*.

```
function f = regorder(x, t, Ca0, Ca)
% x(1)=k', x(2)=alpha
n = length(t);
for i = 1:n
    f(i,1) = x(1)*(1-x(2))*t(i) - Ca0^(1-x(2)) + Ca(i)^(1-x(2));
end
```

The built-in function *fsolve* is used to get results:

```

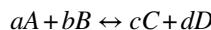
>> t = [0 50 100 150 200 250 300];
>> Ca = [0.05 0.038 0.0306 0.0256 0.0222 0.0195 0.0174]; Ca0 = 0.05;
>> x = fsolve(@regorder,[0.2 2],[],t,Ca0,Ca)
Warning: Trust-region-dogleg algorithm of FSOLVE cannot handle non-square
systems; using Levenberg-Marquardt algorithm instead.
> In fsolve at 285
No solution found.
fsolve stopped because the last step was ineffective. However, the vector of
function
values is not near zero, as measured by the default value of the function
tolerance.
<stopping criteria details>
x =
0.1449    2.0412

```

We can see that $k' = 0.1449 \text{ dm}^3/(\text{mol} \cdot \text{min})$ and $\alpha = 2.0412$.

6.1.2 REACTION EQUILIBRIUM

The constant representing chemical equilibrium is defined in terms of the activities of the species. The equilibrium constant for the reaction



is expressed as

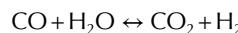
$$K = \frac{a_C^c a_D^d}{a_A^a a_B^b}$$

Since the activity is the fugacity divided by the fugacity of the standard state, the activity is the fugacity in the gas-phase reaction. When the pressure is not high (usually less than 10 atm), the gas can be considered as ideal gas and the equilibrium constant can be represented in terms of mole fractions (y_i) of the species:

$$K = \frac{y_C^c y_D^d}{y_A^a y_B^b} P^{c+d-a-b}$$

Example 6.3: Equilibrium of Water–Gas Shift Reaction³

Consider the water–gas shift reaction to make hydrogen for fuel cell applications:



It was found that the value of the equilibrium constant $K = 148.4$ at 500 K. The reaction feed consists of 1 mol of CO and 1 mol of H₂O. Determine compositions at equilibrium.

Solution

$$K = 148.4 = \frac{y_{\text{CO}_2} y_{\text{H}_2}}{y_{\text{CO}} y_{\text{H}_2\text{O}}}.$$

Let x be the number of moles consumed in the reaction. Then we have

$$K = 148.4 = \frac{(x/2)(x/2)}{[(1-x)/2][(1-x)/2]} = \frac{x^2}{(1-x)^2}$$

x is obtained from the solution of the nonlinear equation

$$f(x) = 148.4 - \frac{x^2}{(1-x)^2} = 0$$

```
>> f = @(x) 148.4-x^2/(1-x)^2;
>> x0 = 0.5; x = fzero(f,x0)
x =
0.9241
```

6.1.3 REACTION CONVERSION

For batch reaction systems, the relation between the reaction conversion and the concentration is given by

$$C_A = \frac{N_A}{V} = \frac{N_{A0}(1-X)}{V}$$

For constant reactor volume ($V = V_0$),

$$C_A = \frac{N_{A0}(1-X)}{V_0} = C_{A0}(1-X)$$

For flow reaction systems, the concentration is represented as

$$C_A = \frac{F_A}{v} = \frac{F_{A0}(1-X)}{v}$$

$v = v_0$ for liquid-phase reaction, and we have

$$C_A = \frac{F_{A0}(1-X)}{v_0} = C_{A0}(1-X)$$

For gas-phase reaction, v can be represented as

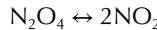
$$v = v_0(1+\epsilon X) \frac{P_0}{P} \frac{T}{T_0}$$

and we have

$$C_A = \frac{F_{A0}(1-X)}{v} = \frac{F_{A0}(1-X)}{v_0(1+\epsilon X)} \frac{P_0}{P} \frac{T_0}{T} = C_{A0} \left(\frac{1-X}{1+\epsilon X} \right) \frac{P}{P_0} \frac{T_0}{T}$$

Example 6.4: Equilibrium Conversion⁴

The reversible gas-phase decomposition of nitrogen tetroxide (N_2O_4) to nitrogen dioxide,



is to be carried out at constant temperature of 340 K. The feed consists of pure N_2O_4 at 202.6 kPa (2 atm). The concentration equilibrium constant, K_C , at 340 K is 0.1 mol/dm³ and the rate constant is $K_{\text{N}_2\text{O}_4} = 0.5 \text{ min}^{-1}$. Calculate the equilibrium conversion of N_2O_4 in a flow reactor. The equilibrium constant is given by

$$K_C = \frac{C_{Be}^2}{C_{Ae}}$$

Solution

For flow system,

$$K_C = \frac{C_{Be}^2}{C_{Ae}} = \frac{\left[2C_{A0}X_e/(1+\epsilon X_e)\right]^2}{C_{A0}(1-X_e)/(1+\epsilon X_e)} = \frac{4C_{A0}X_e^2}{(1-X_e)/(1+\epsilon X_e)}$$

Rearrangement of this equation yields a nonlinear equation

$$f(X_e) = 4C_{A0}X_e^2 - K_C(1-X_e)(1+\epsilon X_e) = 0$$

where $C_{A0} = \frac{y_{A0}P_0}{RT_0}$, $y_{A0} = 1$, $P_0 = 2 \text{ atm}$, $R = 0.082 \text{ atm}\cdot\text{dm}^3/(\text{mol}\cdot\text{K})$, $T_0 = 340 \text{ K}$, and $\epsilon = y_{A0}\delta = 1(2-1) = 1$.

The equilibrium conversion can be determined by using the built-in function *fzero*:

```
>> P = 2; T = 340; R = 0.082; Kc = 0.1; ya0 = 1; epsilon = 1;
>> Ca0 = ya0*P/(R*T); x0 = 0.5;
>> fF = @(x) 4*Ca0*x^2 - Kc*(1-x)*(1+epsilon*x);
>> Xe = fzero(fF,x0)
Xe =
0.5084
```

6.1.3.1 Estimation of Reaction Rate Parameters Using Conversion

The order of reaction and corresponding reaction rate constant can be estimated using measured data for concentration and conversion. For a decomposition reaction of reactant A carried out in a continuous-stirred tank reactor (CSTR), the relation between the flow rate of reactant, v_0 , and the conversion of A, X_A , may be represented as

$$v_0 = \frac{kVC_{A0}^{n-1}(1+X_A)^n}{X_A(1+\epsilon X_e)^n}$$

where

k is the reaction rate constant

C_{A0} is the initial concentration of A

V is the reactor volume

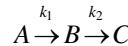
This equation can be rewritten as

$$\ln\left(\frac{v_0 C_{A0} X_A}{V}\right) = \ln(k) + n \ln\left(\frac{C_{A0}(1+X_A)}{1+\epsilon X_A}\right)$$

This is a typical linear relation of the form $z = \beta x + \gamma y$.

6.1.4 SERIES REACTIONS

Consider the reaction sequence in which species B is the desired product:



Reaction rates can be represented as

$$\frac{dC_A}{dt} = -k_1 C_A$$

$$\frac{dC_B}{dt} = k_1 C_A - k_2 C_B$$

$$\frac{dC_C}{dt} = k_2 C_B$$

A is fed into a batch reactor and the initial concentration of A is C_{A0} . Part of A is converted to the desired product B . If the reaction is allowed to proceed for a long time in the batch reactor, part of B will be converted to the undesired product C . Integration of the rate equations with the initial conditions $C_A = C_{A0}$, $C_B = C_C = 0$ at $t = 0$ gives concentrations of each species as

$$C_A = C_{A0} e^{-k_1 t}$$

$$C_B = C_{A0} \frac{k_1}{k_2 - k_1} (e^{-k_1 t} - e^{-k_2 t})$$

$$C_C = C_{A0} - C_A - C_B$$

For series reactions carried out in a batch or a plug-flow reactor, the maximum possible concentration of B is given by

$$C_{Bmax} = C_{A0} \left(\frac{k_1}{k_2}\right)^{k_2/(k_2 - k_1)}$$

C_{Bmax} is reached at

$$t_{max} = \frac{\ln(k_2/k_1)}{k_2 - k_1}$$

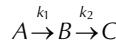
The reactor volume is given by

$$V = vt$$

where v is the volumetric flow rate.

Example 6.5: Series Reactions

The elementary liquid-phase series reaction is carried out in a batch reactor:



In this reaction, A is decomposed to the desired product B and the flow rate of the feed containing A is 25 liter/min. Determine the maximum concentration of B and the time when the concentration of B reaches the maximum value. The initial concentrations of A , B , and C are $C_{A0} = 2.5$ mol/liter, $C_B = C_C = 0$. The reaction rate constants are $k_1 = 0.2 \text{ min}^{-1}$, $k_2 = 0.1 \text{ min}^{-1}$.

Solution

The script *serrb* integrates differential equations using the built-in function *ode45* and plots concentrations versus time.

```
% serrb.m
v = 25; k1 = 0.2; k2 = 0.1; Ca0 = 2.5; Cb0 = 0; Cc0 = 0;
tspan = 0:0.01:15; C0 = [Ca0 Cb0 Cc0];
dCdt = @(t,C) [-k1*C(1); k1*C(1)-k2*C(2); k2*C(2)];
[t,C]=ode45(dCdt, tspan, C0);
plot(t,C(:,1),'-',t,C(:,2),':',t,C(:,3), '--');
xlabel('Time(min)'), ylabel('Concentration(mol/liter)'), legend('A','B','C');
Cmax = max(C(:,2)), tmax = t(find(C(:,2) == Cmax)), Vol = v*tmax
```

The script *serrb* produces the following results and generates the concentration profiles as shown in Figure 6.2:

```
>> serrb
Cmax =
1.2500
tmax =
6.9300
Vol =
173.2500
```

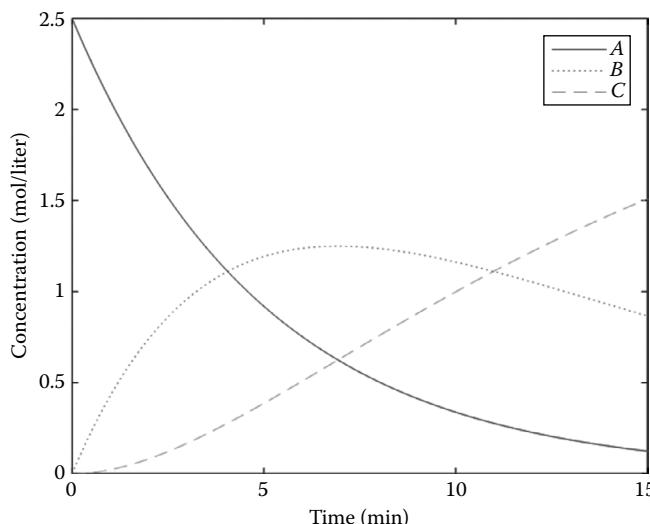


FIGURE 6.2 Concentration profiles in series reactions.

6.2 CONTINUOUS-STIRRED TANK REACTOR (CSTR)

6.2.1 CONCENTRATION CHANGES WITH TIME

The elementary irreversible liquid-phase reaction



is carried out in a series of three CSTRs as shown in [Figure 6.3](#). The volume of each reactor is V_i ($i = 1, 2, 3$), the volumetric flow rate of each stream is v_i ($i = 1, 2, 3$) (dm^3/min), and the concentration of species i in the reactor j is C_{ij} ($i = A, B; j = 1, 2, 3$) (gmol/dm^3).

For a liquid-phase reaction, the volume change with reaction may be neglected. The material balances on each reactor yields the following ordinary differential equations:

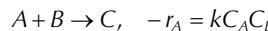
$$\frac{dC_{A1}}{dt} = \frac{1}{V_1} (v_{0A}C_{A0} - v_1C_{A1} - kV_1C_{A1}C_{B1}), \quad \frac{dC_{B1}}{dt} = \frac{1}{V_1} (v_{0B}C_{B0} - v_1C_{B1} - kV_1C_{A1}C_{B1})$$

$$\frac{dC_{A2}}{dt} = \frac{1}{V_2} (v_1C_{A1} - v_2C_{A2} - kV_2C_{A2}C_{B2}), \quad \frac{dC_{B2}}{dt} = \frac{1}{V_2} (v_1C_{B1} - v_2C_{B2} - kV_2C_{A2}C_{B2})$$

$$\frac{dC_{A3}}{dt} = \frac{1}{V_3} (v_2C_{A2} - v_3C_{A3} - kV_3C_{A3}C_{B3}), \quad \frac{dC_{B3}}{dt} = \frac{1}{V_3} (v_2C_{B2} - v_3C_{B3} - kV_3C_{A3}C_{B3})$$

Example 6.6: CSTRs in Series⁵

The elementary irreversible liquid-phase reaction



is carried out in a series of three identical CSTRs. The initial concentrations of A and B are $C_{A0} = C_{B0} = 2 \text{ gmol}/\text{dm}^3$, the inlet flow rates of A and B are $v_{0A} = v_{0B} = 6 \text{ dm}^3/\text{min}$, the reaction rate constant is $k = 0.5$, the volume of each reactor is $V_1 = V_2 = V_3 = 200 \text{ dm}^3$, and the flow rate of each stream is $v_1 = v_2 = v_3 = 12 \text{ dm}^3/\text{min}$. Plot the concentration of A exiting each reactor, C_{Ai} ($i = 1, 2, 3$), during start-up to the final time $t = 20$ ($0 \leq t \leq 20$).

Solution

The initial concentrations of A and B are C_{A0} and C_{B0} , respectively, only in the first reactor and are zero in the remaining reactors. In the system of differential equations defined in the script `ser3v`,

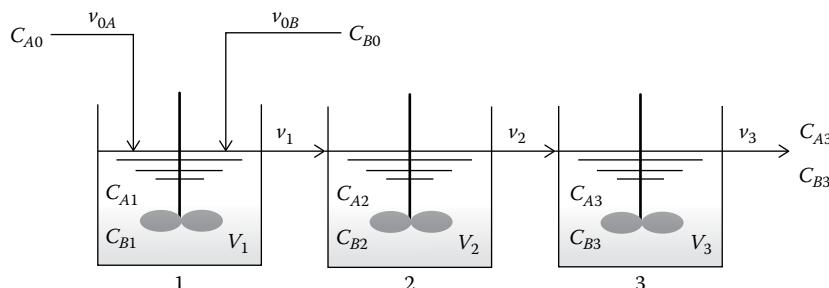


FIGURE 6.3 CSTRs in series.

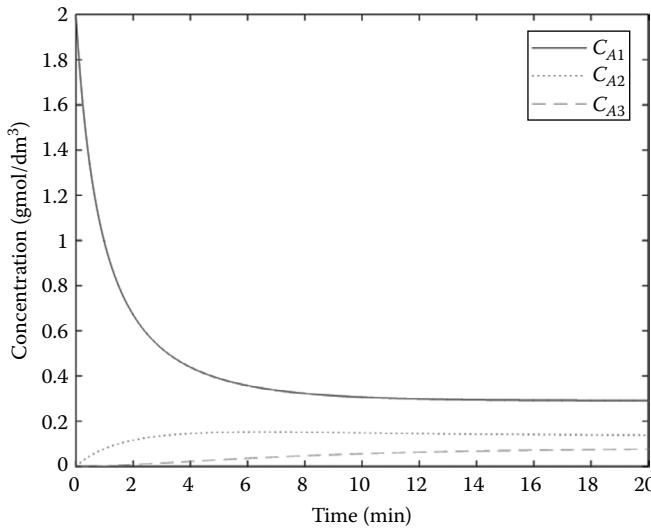


FIGURE 6.4 Concentration profiles in the train of three CSTRs.

$C(1) = C_{A1}$, $C(3) = C_{A2}$, $C(5) = C_{A3}$, $C(2) = C_{B1}$, $C(4) = C_{B2}$, and $C(6) = C_{B3}$. The script *ser3v* uses the built-in function *ode45* to calculate concentrations.

```
% ser3v.m
v0 = 6; v = 12; V = 200; Ca0 = 2; Cb0 = 2; k = 0.5;
tspan = 0:0.01:20; C0 = [Ca0 Cb0 0 0 0 0];
dCdt = @(t,C) [(v0*Ca0-v*C(1)-k*V*C(1)*C(2))/V;
    (v0*Cb0-v*C(2)-k*V*C(1)*C(2))/V;
    (v*C(1)-v*C(3)-k*V*C(3)*C(4))/V;
    (v*C(2)-v*C(4)-k*V*C(3)*C(4))/V;
    (v*C(3)-v*C(5)-k*V*C(5)*C(6))/V;
    (v*C(4)-v*C(6)-k*V*C(5)*C(6))/V];
[t,C]=ode45(dCdt, tspan, C0);
plot(t,C(:,1),'-',t,C(:,3),':',t,C(:,5), '--');
xlabel('Time(min)'), ylabel('Concentration(gmol/dm^3)')
legend('C_{A1}', 'C_{A2}', 'C_{A3}')
```

The script *ser3v* produces the concentration profiles shown in [Figure 6.4](#):

```
>> ser3v
```

6.2.2 EXOTHERMIC REACTION

[Figure 6.5](#) shows a well-mixed, constant-volume CSTR. A single 1st-order exothermic reaction $A \rightarrow B$ is carried out in the reactor. To remove the heat of reaction, the reactor is surrounded by a jacket through which a cooling liquid flows. Let's assume that the heat losses to the surroundings are negligible and that the densities and heat capacities of the reactants and products are both equal and constant.

The reaction rate is given by

$$-r_A = kC_A, \quad k = \alpha e^{-E/RT}$$

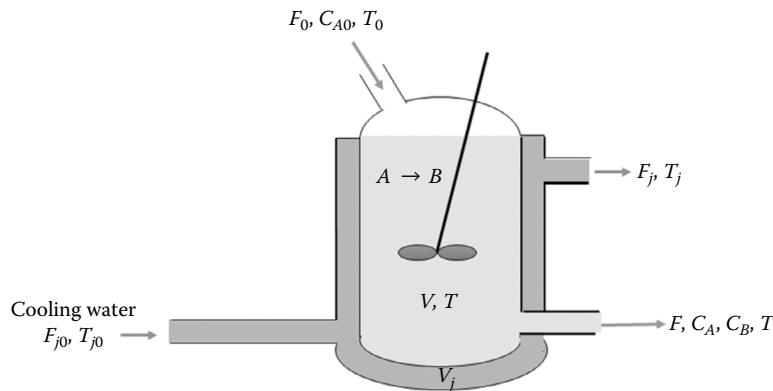


FIGURE 6.5 A CSTR with a cooling jacket.

The rate of heat transferred from the reactor to the cooling jacket is given by

$$Q = UA(T - T_j)$$

where

U is the overall heat transfer coefficient

A is heat transfer area

T is the *temperature* of the reactor

T_j is the temperature of the cooling jacket

The mass balance for the species A yields

$$F_0 C_{A0} - F C_A - k V C_A = 0 \quad \text{or} \quad C_A = \frac{F_0 C_{A0}}{F + k V}$$

From the energy balance on the reactor, we have

$$\rho C_p (F_0 T_0 - F T) - \lambda k V C_A - U A (T - T_j) = 0$$

The energy balance on the cooling jacket gives

$$\rho_j C_j F_j (T_{j0} - T_j) + U A (T - T_j) = 0 \quad \text{or} \quad T_j = \frac{\rho_j C_j F_j T_{j0} + U A T}{\rho_j C_j F_j + U A}$$

where

λ is the heat of reaction

ρ and ρ_j are the densities of reactants and cooling liquid, respectively

C_p and C_j are the heat capacities of reactants and cooling liquid, respectively

Example 6.7: Nonisothermal CSTR

The liquid-phase 1st-order irreversible exothermic reaction $A \rightarrow B$ is carried out in a CSTR. The reaction rate is given by $-r_A = k C_A$, $k = \alpha e^{-E/RT}$.

1. Calculate the steady-state values of C_A , T , and T_j .
2. This exothermic reaction system may exhibit multiple steady states. The energy balance on the reactor yields

$$\rho C_p (F_0 T_0 - FT) - \lambda k V C_A - UA(T - T_j) = 0$$

Substituting $k = \alpha e^{-E/RT}$, $C_A = \frac{F_0 C_{A0}}{F + kV}$, and $T_j = \frac{\rho_j C_j F_j T_{j0} + UAT}{\rho_j C_j F_j + UA}$ into this equation gives

$$f(T) = \rho C_p (F_0 T_0 - FT) - \frac{F_0 C_{A0} V \lambda \alpha e^{-E/RT}}{F + \alpha e^{-E/RT} V} - UA \rho_j C_j F_j \left(\frac{T - T_{j0}}{\rho_j C_j F_j + UA} \right) = 0$$

Solve the nonlinear equation $f(T) = 0$ and plot $f(T)$ versus T to verify multiple steady states.

Flow rate of the reactant feed:	$F_0 = 40 \text{ ft}^3/\text{hr}$
Flow rate of the product stream:	$F = 40 \text{ ft}^3/\text{hr}$
Initial concentration of the species A:	$C_{A0} = 0.55 \text{ lbmol/ft}^3$
Reactor volume:	$V = 48 \text{ ft}^3$
Flow rate of the cooling water:	$F_{j0} = F_j = 49.9 \text{ ft}^3/\text{hr}$
Heat capacity of the reactant:	$C_p = 0.75 \text{ btu/lb}_m/\text{R}$
Heat capacity of the cooling water:	$C_j = 1 \text{ btu/lb}_m/\text{R}$
Rate constant parameter:	$\alpha = 7.08 \times 10^{10} \text{ hr}^{-1}$
Activation energy:	$E = 30,000 \text{ btu/lbmol}$
Density of the reactant:	$\rho = 50 \text{ lb}_m/\text{ft}^3$
Density of the cooling water:	$\rho_j = 62.3 \text{ lb}_m/\text{ft}^3$
Overall heat transfer coefficient:	$U = 150 \text{ btu}/(\text{hr} \cdot \text{ft}^2 \cdot ^\circ\text{R})$
Heat transfer area:	$A = 250 \text{ ft}^2$
Inlet temperature of the cooling water:	$T_{j0} = 530 \text{ }^\circ\text{R}$
Inlet temperature of the reactant:	$T_0 = 530 \text{ }^\circ\text{R}$
Heat of the reaction:	$\lambda = -30,000 \text{ btu/lbmol}$
Volume of the cooling jacket:	$V_j = 12 \text{ ft}^3$
Gas constant:	$R = 1.9872 \text{ btu/lbmol}/\text{R}$

Solution

1. We can formulate the mass and energy balances that apply to the reactor and the cooling jacket. Required steady-state values can be obtained from the solution of the system of nonlinear equations consisting of these balances. Rearrangement of the mass and energy balances yields the following nonlinear equations ($x_1 = C_A$, $x_2 = T$, $x_3 = T_j$):

Mass balance on the reactor: $f_1 = F_0 C_{A0} - F x_1 - \alpha V e^{-E/R} x_2 x_1 = 0$

Energy balance on the reactor: $f_2 = \rho C_p (F_0 T_0 - F x_2) - \lambda \alpha V e^{-E/R} x_2 x_1 - U A (x_2 - x_3) = 0$

Energy balance on the cooling jacket: $f_3 = \rho_j C_j (T_{j0} - x_3) + U A (x_2 - x_3) = 0$

Set the initial estimates of the final solution as $x_0 = [C_{A0} \ T_0 \ T_{j0}]$. The MATLAB® script exocstr uses the built-in function *fsove* to give the desired results.

```
% exocstr.m
% Data:
F0 = 40; F = 40; Fj = 49.9; Ca0=0.55; V = 48;
rho = 50; rhoj = 62.3; Cp = 0.75; Cj = 1; A = 250; U = 150;
T0 = 530; Tj0 = 530; alp=7.08e10; lam = -3e4; E = 3e4; R = 1.9872;
% Define nonlinear equations
fun = @(x) [F0*Ca0 - F*x(1) - alp*V*x(1)*exp(-E/R*x(2));
            rho*Cp*(F0*T0-F*x(2))-lam*alp*V*x(1)*exp(-E/R*x(2))-U*A*(x(2) - x(3));
            rhoj*Cj*Fj*(Tj0 - x(3)) + U*A*(x(2) - x(3))];
```

```
% Solution of nonlinear equation system
x0 = [Ca0 T0 Tj0]; % initial guess
x = fsolve(fun, x0);
Ca = x(1), T = x(2), Tj = x(3)
```

The script *exocstr* produces the following outputs:

```
>> exocstr
Ca =
0.5214
T =
537.8548
Tj =
537.2534
```

2. The nonlinear equation

$$f(T) = \rho C_p (F_0 T_0 - FT) - \frac{F_0 C_{A0} V \lambda \alpha e^{-E/RT}}{F + \alpha e^{-E/RT} V} - UA \rho_j C_j F_j \left(\frac{T - T_{j0}}{\rho_j C_j F_j + UA} \right) = 0$$

is solved by the script *exocstr2*, which also generates a plot $f(T)$ versus T .

```
% exocstr2.m
% Data:
F0 = 40; F = 40; Fj = 49.9; Ca0=0.55; V = 48;
rho = 50; rhoj = 62.3; Cp = 0.75; Cj = 1; A = 250; U = 150;
T0 = 530; Tj0 = 530; alp=7.08e10; lam = -3e4; E = 3e4; R = 1.9872;
% Define nonlinear functions
fT = @(T) [rho*Cp*(F0*T0 - F*T) - (F0*Ca0*V*lam*alp*exp(-E/R./T)) ./. . .
(F+alp*V*exp(-E/R./T)) - U*A*rhoj*Cj*Fj*(T-Tj0) ./ (rhoj*Cj*Fj + U*A)];
% Solve nonlinear equations
T0 = T0+150 % initial guess
x = fzero(fT, T0)
% Plot of f(T) versus T
Tv = 500:0.1:700; Fv = fT(Tv); Fv0 = Fv*0;
plot(Tv,Fv,Tv,Fv0), xlabel('T(R)'), ylabel('f(T)')
```

Different steady states can be determined by solving the nonlinear equation with different initial estimates of the final solution. The script *exocstr2* computes T for three different initial estimates of $T_0 = 530, 580, 680$ (°R) and generates a plot of $f(T)$ as shown in [Figure 6.6](#).

```
>> exocstr2
T0 =
530
x =
537.8548
>> exocstr2
T0 =
580
x =
590.3498
>> exocstr2
T0 =
680
x =
671.2783
```

We can see that the reaction system exhibits three different steady states of $T_s = 537.8548$, 590.3498 , 671.2783 (°R).

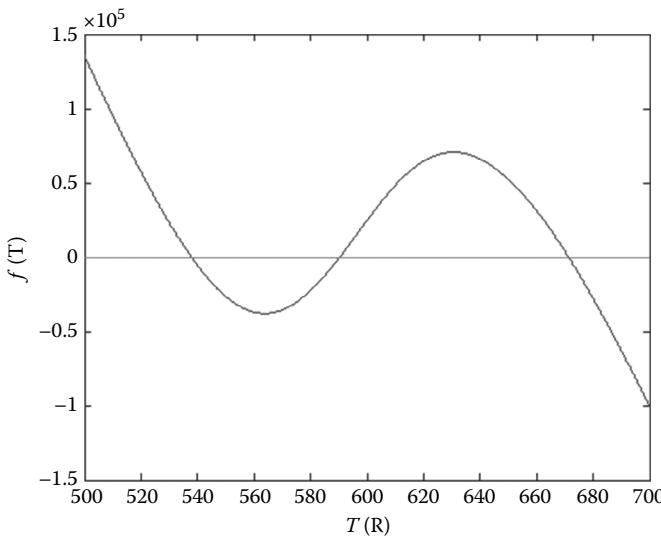


FIGURE 6.6 Multiple steady-state solutions for the exothermic reaction system.

6.2.3 MULTIPLE REACTIONS IN A CSTR

For q liquid-phase reactions occurring where N different species are present, the mass balance on each reactor yields⁶

$$F_{10} - F_1 = -r_1 V = V \sum_{i=1}^q (-r_{i1}) = V f_1(C_1, \dots, C_N)$$

$$F_{20} - F_2 = -r_2 V = V \sum_{i=1}^q (-r_{i2}) = V f_2(C_1, \dots, C_N)$$

⋮

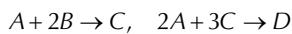
$$F_{j0} - F_j = -r_j V = V \sum_{i=1}^q (-r_{ij}) = V f_j(C_1, \dots, C_N)$$

⋮

$$F_{N0} - F_N = -r_N V = V \sum_{i=1}^q (-r_{iN}) = V f_N(C_1, \dots, C_N)$$

Example 6.8: Multiple Reactions in a Liquid-Phase CSTR⁷

The following liquid-phase reactions take place in a 2500 dm³ CSTR:



The reaction rate for each species is expressed as

$$r_A = -k_a C_A C_B^2 - \frac{2}{3} k_c C_A^2 C_C^3, \quad r_B = -2k_a C_A C_B^2,$$

$$r_C = k_a C_A C_B^2 - k_c C_A^2 C_C^3, \quad r_D = \frac{1}{3} k_c C_A^2 C_C^3$$

Determine the concentrations of A , B , C , and D exiting the reactor, along with the exiting selectivity, which is defined as $S_{C/D} = \frac{C_C}{C_D}$.

Data: $k_a = 10 \text{ (dm}^3/\text{mol})^2/\text{min}$, $k_c = 15 \text{ (dm}^3/\text{mol})^4/\text{min}$, $v_0 = 100 \text{ dm}^3/\text{min}$, $C_{A0} = C_{B0} = 2 \text{ mol/dm}^3$

Solution

The mole balance for each species gives

$$f(C_A) = v_0 C_{A0} - v_0 C_A + r_A V = 0, \quad f(C_B) = v_0 C_{B0} - v_0 C_B + r_B V = 0,$$

$$f(C_C) = -v_0 C_C + r_C V = 0, \quad f(C_D) = -v_0 C_D + r_D V = 0$$

The system of nonlinear equations is defined by the MATLAB® function *cstrmult*. The dependent variable vector C is defined as $C^T = [C_A \ C_B \ C_C \ C_D]$.

```
function fC = cstrmult(C,ka,kc,Ca0,Cb0,v0,V)
% C(1)=Ca, C(2)=Cb, C(3)=Cc, C(4)=Cd
ra = -ka*C(1)*C(2)^2 - 2*kc*C(1)^2*C(3)^3/3;
rb = -2*ka*C(1)*C(2)^2;
rc = ka*C(1)*C(2)^2 - kc*C(1)^2*C(3)^3;
rd = kc*C(1)^2*C(3)^3/3;
fC = [v0*Ca0 - v0*C(1) + ra*V;
      v0*Cb0 - v0*C(2) + rb*V;
      -v0*C(3) + rc*V;
      -v0*C(4) + rd*V];
end
```

The script *usecstrmult* calls the function *cstrmult* and employs the built-in function *fsolve* to solve the system of nonlinear equations. In the calculation of the selectivity $S_{C/D}$, the value of the selectivity is set to zero when C_D is very small (remember that the initial value of C_D is 0).

```
% usecstrmult.m
clear all;
ka = 10; kc = 15; V = 2500; v0 = 100; Ca0 = 2; Cb0 = 2;
C0 = [Ca0 Cb0 0 0];
[C fval] = fsolve(@cstrmult,C0,[],ka,kc,Ca0,Cb0,v0,V);
Ca = C(:,1); Cb = C(:,2); Cc = C(:,3); Cd = C(:,4);
n = length(Cd); Scd = zeros(1,n);
for i = 1:n
    if Cd(i) <= 1e-4
        Scd(i) = 0;
    else
        Scd(i) = Cc(i)/Cd(i);
    end
end
fprintf('The final concentration of each species: \n');
fprintf('Caf=%g, Cbf=%g, Ccf=%g, Cdf=%g\n',Ca(end),Cb(end),Cc(end),Cd(end));
fprintf('The final selectivity: Scdf = %g\n',Scd(end));
```

The script *usecstrmult* generates the following outputs:

```
>> usecstrmult
Equation solved.
fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.
<stopping criteria details>
The final concentration of each species:
Caf=0.532653, Cbf=0.0848008, Ccf=0.192978, Cdf=0.254874
The final selectivity: Scdf = 0.757153
```

6.3 BATCH REACTOR

6.3.1 ESTIMATION OF BATCH REACTION PARAMETERS

A conversion reaction of *A* is carried out in a batch reactor. A mass balance on the batch reactor yields

$$\frac{dC_A}{dt} = -kC_A^n$$

If the measured data on the concentration of *A* (C_A) with respect to time t are available, the reaction order and rate constant can be determined by applying the differential method. Taking a logarithm on the rate equation yields

$$\ln\left(-\frac{dC_A}{dt}\right) = \ln(k) + n \ln(C_A)$$

Thus, determination of the reaction order n and rate constant k becomes a typical linear regression problem.

Example 6.9: Reaction Parameters in a Batch Reactor⁸

The liquid-phase bromination of xylene at 17°C is carried out in a batch reactor. Iodine is used as a catalyst, and small quantities of the reactant bromine are introduced into the reactor containing the reactant xylene in considerable excess. The concentrations of the reactant xylene and the iodine catalyst are approximately constant during the reaction. A mass balance on the batch reactor yields

$$\frac{dC_{Br_2}}{dt} = -kC_{Br_2}^n$$

where

C_{Br_2} is the concentration of bromine (gmol/dm³)

k is a *pseudo* rate constant that depends on the iodine and xylene concentrations

n is the *reaction order*

Data on the concentration of bromine (C_{Br_2}) are shown in Table 6.2. Estimate the rate constant k and the reaction order n .

Solution

From the rate equation $\frac{dC_{Br_2}}{dt} = -kC_{Br_2}^n$ (C_{Br_2} : concentration of bromine), we have

$$\ln\left(-\frac{dC_{Br_2}}{dt}\right) = \ln(k) + n \ln(C_{Br_2})$$

TABLE 6.2
Concentration of Bromine versus Time

<i>t</i> (min)	<i>C_{Br}</i> (gmol/dm ³)	<i>t</i> (min)	<i>C_{Br}</i> (gmol/dm ³)
0	0.3335	19.60	0.1429
2.25	0.2965	27.00	0.1160
4.50	0.2660	30.00	0.1053
6.33	0.2450	38.00	0.0830
8.00	0.2255	41.00	0.0767
10.25	0.2050	45.00	0.0705
12.00	0.1910	47.00	0.0678
13.50	0.1794	57.00	0.0553
15.60	0.1632	63.00	0.0482
17.85	0.1500		

The built-in function *diff* can be used to execute the numerical differentiation of data. Then the linear regression method is employed to calculate *n* and *k*. The relation $x = (A^T A)^{-1} A^T b$ can be used in the linear regression where $x(1) = n$ and $x(2) = \ln(k)$. The script *rnk* performs the calculation procedure.

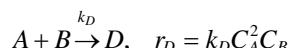
```
% rnk.m
t = [0 2.25 4.50 6.33 8.00 10.25 12.00 13.50 15.60 17.85 ...
      19.60 27.00 30.00 38.00 41.00 45.00 47.00 57.00 63.00];
Cb = [0.3335 0.2965 0.2660 0.2450 0.2255 0.2050 0.1910 0.1794 0.1632 ...
      0.1500 0.1429 0.1160 0.1053 0.0830 0.0767 0.0705 0.0678 0.0553 0.0482];
dCb = diff(Cb) ./ diff(t); dCb = [dCb dCb(end)];
n = length(t); A = [(log(Cb))' ones(n,1)];
x = (A'*A)^-1 * A' * (log(-dCb))'
n = x(1), k = exp(x(2))
```

The script *rnk* produces the following outputs:

```
>> rnk
x =
    1.5128
   -2.4454
n =
    1.5128
k =
    0.0867
```

6.3.2 SEMIBATCH REACTORS

One of the best reasons to use semibatch reactors is to enhance selectivity in liquid-phase reactions.⁹ For example, consider the following two simultaneous reactions being carried out in a semibatch reactor. The desired product is *D*:



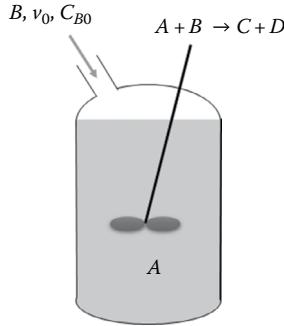
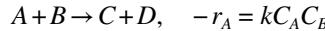


FIGURE 6.7 Semibatch reactor.

the instantaneous selectivity $S_{D/U}$ is the ratio of these two rates:

$$S_{D/U} = \frac{r_D}{r_U} = \frac{k_D C_A^2 C_B}{k_U C_A C_B^2} = \frac{k_D}{k_U} \frac{C_A}{C_B}$$

Consider a semibatch reactor that is charged with pure A and to which B is fed slowly to A as shown in Figure 6.7. The elementary liquid-phase reaction is carried out in the reactor¹⁰:



Mole balance on each species yields

$$\frac{dC_A}{dt} = r_A - \frac{v_0 C_A}{V}, \quad \frac{dC_B}{dt} = \frac{v_0 (C_{B0} - C_B)}{V} + r_B, \quad \frac{dC_C}{dt} = r_C - \frac{v_0 C_C}{V}, \quad \frac{dC_D}{dt} = r_D - \frac{v_0 C_D}{V}$$

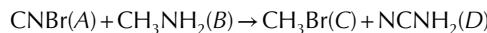
$$-r_A = -r_B = r_C = r_D$$

The semibatch reactor volume can be expressed as a function of time, and the conversion of species A is represented in terms of the reactor volume V as

$$V = V_0 + v_0 t, \quad X = \frac{C_{A0}V_0 - C_A V}{C_{A0}V_0}$$

Example 6.10: Semibatch Reactor¹¹

Methyl bromide is produced by the irreversible liquid-phase reaction



The reaction is carried out isothermally in a semibatch reactor. The reaction rate is given by $-r_A = k C_A C_B$ where $k = 2.2 \text{ dm}^3/(\text{sec} \cdot \text{mol})$. The initial volume of liquid in the reactor is $V_0 = 5 \text{ dm}^3$. An aqueous solution of methyl amine (B) at a concentration of $C_{B0} = 0.025 \text{ mol/dm}^3$ is to be fed at a volumetric flow rate of $v_0 = 0.05 \text{ dm}^3/\text{sec}$ to an aqueous solution of bromine cyanide (A) contained in the reactor. The initial concentration of bromine cyanide is $C_{A0} = 0.05 \text{ mol/dm}^3$. Solve for the concentration of each species (A and B) and the rate of reaction as a function of time. Plot the results versus time ($0 \leq t \leq 500 \text{ (sec)}$).

Solution

Since $-r_A = -r_B = r_C = r_D$, we get the following differential equations:

$$\frac{dC_A}{dt} = r_A - \frac{v_0 C_A}{V}, \quad \frac{dC_B}{dt} = \frac{v_0 (C_{B0} - C_B)}{V} + r_A, \quad \frac{dC_C}{dt} = -r_A - \frac{v_0 C_C}{V}, \quad \frac{dC_D}{dt} = -r_A - \frac{v_0 C_D}{V},$$

$$V = V_0 + v_0 t, \quad -r_A = k C_A C_B$$

The MATLAB® function *semibrx* defines these differential equations:

```
function dxdt = semibrx(t,x,k,v0,V0,Cb0)
% x(1)=Ca, x(2)=Cb, x(3)=Cc, x(4)=Cd
rA = -k*x(1)*x(2); V = V0 + v0*t;
dxdt = [rA - v0*x(1)/V;
         rA + (Cb0 - x(2))*v0/V;
         -rA - v0*x(3)/V;
         -rA - v0*x(4)/V];
end
```

The script *usesemibrx* calls the function *semibrx* and uses the built-in function *ode45* to integrate the system of differential equations. The script also calculates the conversion of A as a function of time.

```
% usesemibrx.m
clear all;
k = 2.2; v0 = 0.05; V0 = 5; Cb0 = 0.025; Ca0 = 0.05;
x0 = [Ca0 0 0 0]; tspan = [0 500];
[t x] = ode45(@semibrx,tspan,x0,[],k,v0,V0,Cb0);
Ca = x(:,1); Cb = x(:,2); Cc = x(:,3); Cd = x(:,4);
V = V0 + v0*t; Xa = (Ca0*V0 - Ca.*V)/(Ca0*V0); rA = k*Ca.*Cb;
subplot(1, 2), plot(t,Ca, t,Cb,':', t,Cc,'.-', t,Cd,'--')
xlabel('t(s)'), ylabel('Concentration(mol/dm^3)'), legend('C_A','C_B','C_C','C_D')
subplot(1, 2), plot(t,rA), xlabel('t(s)'), ylabel('Reaction rate(mol/dm^3s)')
```

The script *usesemibrx* generates the plots shown in [Figure 6.8](#):

```
>> usesemibrx
```

6.4 PLUG-FLOW REACTOR

The mass balance for a plug-flow reactor can be expressed as

$$\frac{dX}{dV} = \frac{-r_A}{F_{A0}} \quad \text{or} \quad v_0 \frac{dC_A}{dV} = r_A$$

If the reaction is carried out in isothermal isobaric conditions and the rate of reaction is taken as first order, $F_{A0} = C_{A0}v_0$, $\epsilon = y_0\delta$, and

$$-r_A = kC_A = kC_{A0} \left(\frac{1-X}{1+\epsilon X} \right) \frac{P}{P_0} \frac{T_0}{T} = kC_{A0} \left(\frac{1-X}{1+\epsilon X} \right)$$

Thus, the mass balance equation can be expressed as

$$\frac{dX}{dV} = \frac{k(1-X)}{v_0(1+\epsilon X)}, \quad X|_{V=0} = 0$$

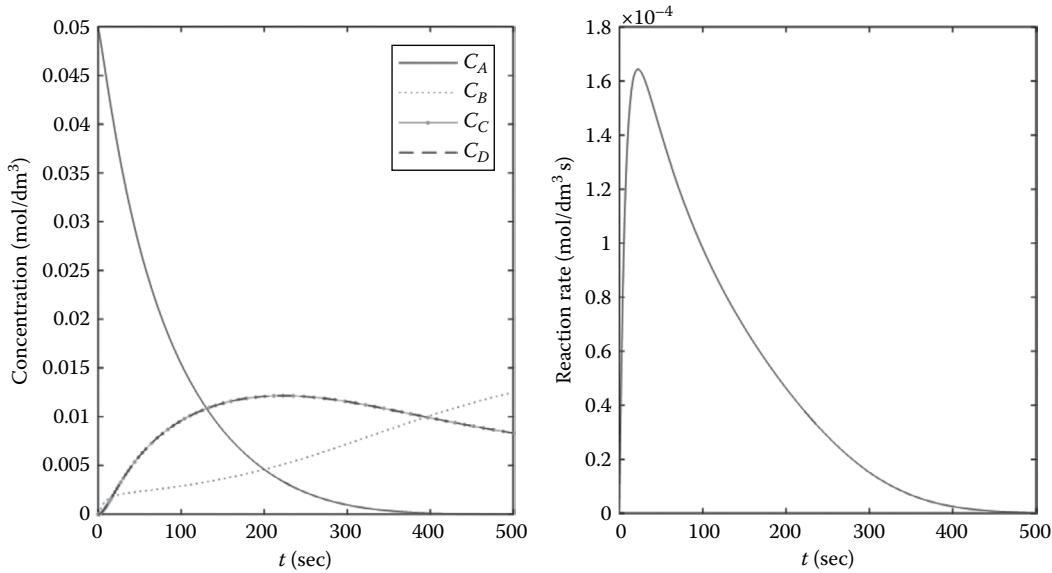


FIGURE 6.8 Concentration and reaction rate versus time.

If the reactant consists of pure A , ϵ is given by $\epsilon = y_0\delta = \delta$. But, if the reactant contains $z\%$ of A , ϵ is given by $\epsilon = y_0\delta = \frac{z\delta}{100}$.

Example 6.11: Isothermal Plug-Flow Reactor¹²

Components A and C are fed to a plug-flow reactor in equimolar amounts, and the reaction $2A \rightarrow B$ takes place in the reactor. The mass balance on each species yields

$$v_0 \frac{dC_A}{dV} = -2kC_A^2, \quad v_0 \frac{dC_B}{dV} = kC_A^2, \quad v_0 \frac{dC_C}{dV} = 0$$

where $v_0 = 0.5$ m/sec and $k = 0.3$ m³/kmol/sec. The initial concentration of each species is $C_{A0} = 2$ kmol/m³, $C_{B0} = 0$ and $C_{C0} = 2$ kmol/m³, and the volume of the reactor represented as the total reactor length is $V_f = 2.4$ m. Plot the concentration change of each species as a function of the reactor volume (represented in length) V .

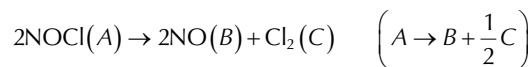
Solution

The differential equations are defined first and the built-in function *ode45* is employed to solve the differential equations. The following code produces the plot shown in Figure 6.9.

```
>> v0 = 0.5; k = 0.3; Vf = 2.4; Vi = [0 Vf]; C0 = [2 0 2];
>> dC = @(V,C) [-2*k*C(1)^2/v0; k*C(1)^2/v0; 0]; [V C] = ode45(dC,Vi,C0);
>> plot(V,C(:,1),V,C(:,2),'-',V,C(:,3), '--'), legend('C_A','C_B','C_C')
>> xlabel('V(length, m)'), ylabel('Concentration(kmol/m^3)')
```

Example 6.12: Decomposition Reaction in a Microreactor¹³

The gas-phase decomposition reaction



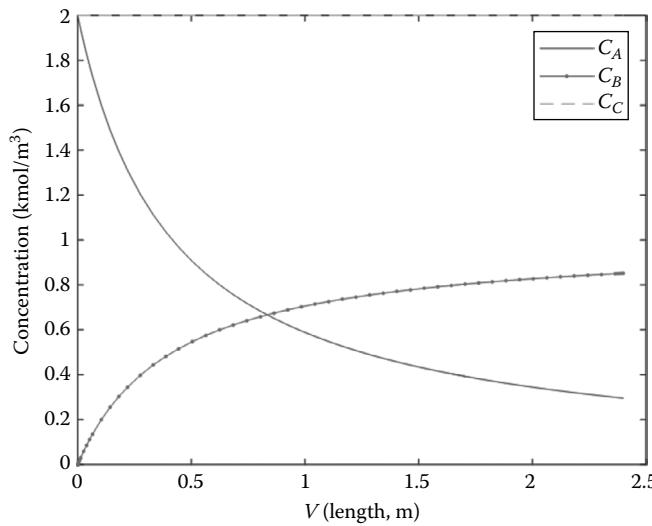


FIGURE 6.9 Concentration changes versus reactor length.

is carried out in a microreactor. The reaction is second order and the reaction rate is given by

$$-r_A = kC_A^2 = kC_{T0}^2 \left(\frac{F_A}{F_T} \right)^2$$

where $C_{T0} = \frac{P}{RT}$, $C_i = C_{T0} \frac{F_i}{F_T}$ ($i = A, B, C$), and $F_T = F_A + F_B + F_C$. Plot the molar flow rate of each species (F_A , F_B , F_C) as a function of reactor volume. The rate constant is given by $k = k_0 \exp\left(\frac{E}{1.987} \left(\frac{1}{500} - \frac{1}{T}\right)\right)$ where $E = 24,000$.

Data: $F_{A0} = 22.6 \text{ } \mu\text{mol/sec} = 2.26 \times 10^{-5} \text{ mol/sec}$, $k_0 = 0.29 \text{ dm}^3/(\text{mol} \cdot \text{sec})$, $R = 8.314 \text{ kPa} \cdot \text{dm}^3/(\text{mol} \cdot \text{K})$, $P = 1641 \text{ kPa}$, $T = 698 \text{ K}$

Solution

$$\frac{dF_i}{dV} = r_i \quad (i = A, B, C)$$

Since $\frac{r_A}{-1} = \frac{r_B}{1} = \frac{r_C}{1/2}$, $r_B = -r_A$, and $r_C = -\frac{1}{2}r_A$. Thus

$$\frac{dF_A}{dV} = -kC_{T0}^2 \left(\frac{F_A}{F_T} \right)^2, \quad \frac{dF_B}{dV} = kC_{T0}^2 \left(\frac{F_A}{F_T} \right)^2, \quad \frac{dF_C}{dV} = \frac{k}{2} C_{T0}^2 \left(\frac{F_A}{F_T} \right)^2$$

Initial conditions are $F_{A0} = 22.6$, $F_{B0} = 0$, $F_{C0} = 0$. Here the integration is performed in the range of $0 \leq V \leq 0.00001$. The function *microrx* defines the differential equations.

```
function dxdv = microrx(v, x, k, Ct0)
% x(1) = Fa, x(2) = Fb, x(3) = Fc
Ft = sum(x);
dxdv = [-k*Ct0^2*(x(1)/Ft)^2;
          k*Ct0^2*(x(1)/Ft)^2;
          (k/2)*Ct0^2*(x(1)/Ft)^2];
end
```

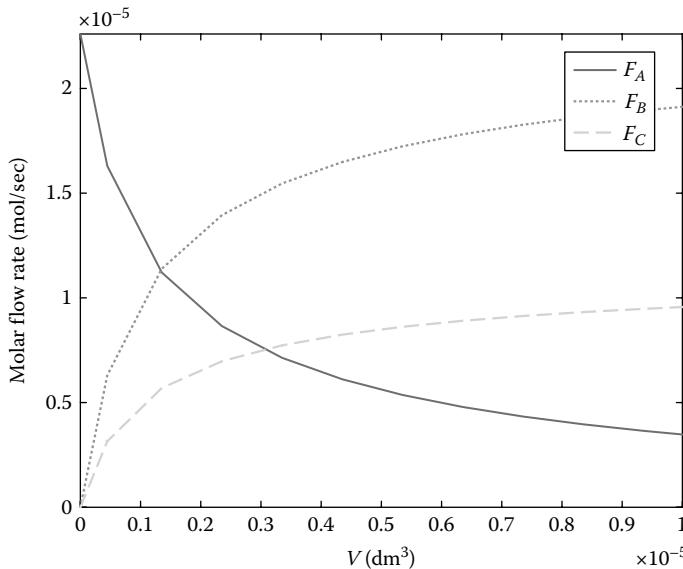


FIGURE 6.10 Profiles of microreactor molar flow rates.

The script *usemicr0rx* calls the function *micr0rx* and uses the built-in solver to integrate the differential equations. The system of differential equations is stiff and we should use the function *ode23s* instead of *ode45*.

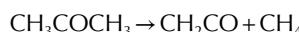
```
% usemicr0rx.m
Fa0 = 2.26e-5; R = 8.314; P0 = 1641; T0 = 698; E = 24000;
Ct0 = P0/(R*T0); Vf = 1e-5; x0 = [Fa0 0 0];
k = 0.29*exp(E/1.987*(1/500-1/T0));
[V x] = ode23s(@micr0rx,[0 Vf],x0,[],k,Ct0);
Fa = x(:,1); Fb = x(:,2); Fc = x(:,3);
fprintf('Final values: Fa = %g, Fb = %g, Fc = %g\n', Fa(end), Fb(end), Fc(end));
plot(V,Fa,V,Fb,'-',V,Fc,'--'), xlabel('V(dm^3)'), axis tight
ylabel('Molar flow rate (mol/sec)'), legend('F_A','F_B','F_C')
```

The script *usemicr0rx* produces the final values and the curves shown in Figure 6.10:

```
>> usemicr0rx
Final values: Fa = 3.47619e-06, Fb = 1.91238e-05, Fc = 9.5619e-06
```

Example 6.13: Cracking of Acetone in a Plug-Flow Reactor¹⁴

The irreversible vapor-phase cracking reaction of acetone (A) to ketene (B) and methane (C)



is carried out adiabatically in a plug-flow reactor. The reaction is first order with respect to acetone and the reaction rate is given by $-r_A = kC_A$ (C_A : concentration of acetone, gmol/m³). From the mass balance equations for the plug-flow reactor, the rate of change of the molar flow rate of each species F_A , F_B , and F_C (gmol/sec) with respect to reactor volume V is given by

$$\frac{dF_A}{dV} = r_A, \quad \frac{dF_B}{dV} = -r_A, \quad \frac{dF_C}{dV} = -r_A$$

The rate constant k (sec⁻¹) can be expressed as a function of temperature T (K):

$$\ln k = 34.34 - \frac{34222}{T}$$

For a gas-phase reactor, the concentration of the acetone C_A (gmol/m³) can be represented as $C_A = \frac{1000y_A P}{8.31T}$. The mole fraction of species i , y_i , is given by $y_i = \frac{F_i}{F_A + F_B + F_C}$ ($i = A, B, C$), and the conversion of acetone can be calculated from $x_A = \frac{F_{A0} - F_A}{F_{A0}}$. An energy balance on a differential volume of the reactor yields

$$\frac{dT}{dV} = \frac{-r_A(-\Delta H)}{F_A C_{pA} + F_B C_{pB} + F_C C_{pC}}$$

where ΔH (J/gmol) is the heat of reaction at temperature T and C_{pi} ($i = A, B, C$) are the molar heat capacities (J/(gmol·K)) of acetone (A), ketene (B), and methane (C) and are given by¹⁵

$$\Delta H = 80,770 + 6.8(T - 298) - 0.00575(T^2 - 298^2) - 1.27 \times 10^{-6}(T^3 - 298^3)$$

$$C_{pA} = 26.2 + 0.183T - 45.86 \times 10^{-6}T^2, \quad C_{pB} = 20.04 + 0.0945T - 30.95 \times 10^{-6}T^2$$

$$C_{pC} = 13.39 + 0.077T - 18.91 \times 10^{-6}T^2$$

The acetone feed flow rate to the reactor is 8000 kg/hr (=38.3 gmol/sec), the inlet temperature is $T = 1150$ K, and the reactor operates at the constant pressure of $P = 162$ kPa (1.6 atm). The volume of the reactor is 4 m³.

1. Calculate the flow rate (gmol/sec) and the mole fraction of each species at the reactor outlet.
2. In order to increase the conversion of acetone, it is suggested to feed nitrogen along with the acetone. The total molar feed rate is maintained constant as 38.3 gmol/sec. Calculate the final conversions and temperatures for the case where 28.3, 18.3, 8.3, 3.3, and 0.0 gmol/sec nitrogen is fed into the reactor and plot the results as a function of reactor volume. The heat capacity of nitrogen is given by

$$C_{pN_2} = 6.25 + 0.00878T - 2.1 \times 10^{-8}T^2.$$

3. Calculate the final conversions and temperatures in the reactor operating at a pressure range of $1.6 \text{ atm} \leq P \leq 5 \text{ atm}$ for acetone feed rates of 10, 20, 30, 35, and 38.3 gmol/sec. The inlet temperature is $T=1035$ K and nitrogen is fed to maintain the total feed rate 38.3 gmol/sec in all cases. Prepare plots of final conversion versus P and F_{A0} and final temperature versus P and F_{A0} .¹⁶

Solution

1. Application of mass and energy balances yields a system of differential equations with four unknown variables (F_A , F_B , F_C , T). The MATLAB® function *adfun* defines the differential equations:

```
function dX = adfun(V,X,PF)
% Differential equations for cracking reaction of acetone
% PF=[P FN2], X(1)== FA, X(2)=FB, X(3)=FC, X(4)=T
P = PF(1); FN2 = PF(2);
T = X(4);
CA = 1000*(X(1)./(X(1)+X(2)+X(3)+FN2))*P/(8.31*T);
```

```

k = exp(34.34 - 34222/T);
dH = 80770 + 6.8*(T-298) - 5.75e-3*(T^2 - 298^2) - 1.27e-6*(T^3 - 298^3);
CpA = 26.2 + 0.183*T - 45.86e-6*T^2;
CpB = 20.04 + 0.0945*T - 30.95e-6*T^2;
CpC = 13.39 + 0.077*T - 18.91e-6*T^2;
CpN2 = 6.25 + 0.00878*T - 2.1e-8*T^2;
rA = -k.*CA;
dX = [rA;
      -rA;
      -rA;
      -rA*(-dH)./(X(1)*CpA + X(2)*CpB + X(3)*CpC + FN2*CpN2)];
end

```

The system of differential equations can be solved by the built-in function *ode45*. The following commands produce the desired outputs:

```

>> pf = [162 0]; Vspan = [0 4]; X0 = [38.3 0 0 1150];
>> [V X] = ode45(@adfun, Vspan, X0, [], pf);
>> FA=X(end,1), FB=X(end,2), FC=X(end,3), T=X(end,4)
FA =
19.9777
FB =
18.3223
FC =
18.3223
T =
920.8778

```

2. When nitrogen is added to the feed stream, the energy balance becomes

$$\frac{dT}{dV} = \frac{-r_A(-\Delta H)}{F_A C_{pA} + F_B C_{pB} + F_C C_{pC} + F_{N_2} C_{pN_2}}$$

and the mole fraction of each species, y_i , is given by

$$y_i = \frac{F_i}{F_A + F_B + F_C + F_{N_2}} \quad (i = A, B, C)$$

These equations are contained in the function *adfun*, which defines the system of differential equations. The following commands compute the final conversions and temperatures corresponding to the flow rates of nitrogen.

```

>> P = 162; Vspan = [0 4]; FN2 = [28.3 18.3 8.3 3.3 0.0]; nF = length(FN2);
>> for i = 1:nF, X0 = [38.3-FN2(i) 0 0 1150]; pf = [P FN2(i)];
>> [V X] = ode45(@adfun, Vspan, X0, [], pf);
>> xc(i) = (X0(1) - X(end,1))/X0(1); Tr(i) = X(end,4); end
>> xc, Tr
xc =
0.5899 0.5183 0.4913 0.4829 0.4784
Tr =
926.3222 921.0767 920.4939 920.6749 920.8778

```

The results are summarized in [Table 6.3](#).

The plots displaying the final conversions and temperatures for the case when the rate of nitrogen is 28.3 gmol/sec can be generated by the following commands (see [Figure 6.11](#)):

```

>> P = 162; Vspan = [0 4]; FN2 = 28.3;
>> X0 = [38.3-FN2 0 0 1150]; pf = [P FN2];
>> [V X] = ode45(@adfun, Vspan, X0, [], pf);

```

TABLE 6.3
Final Conversions and Temperatures

Nitrogen rate (gmol/sec)	28.3	18.3	8.3	3.3	0.0
Final conversion	0.5899	0.5183	0.4913	0.4829	0.4784
Final temperature (K)	926.32	921.08	920.49	920.67	920.88

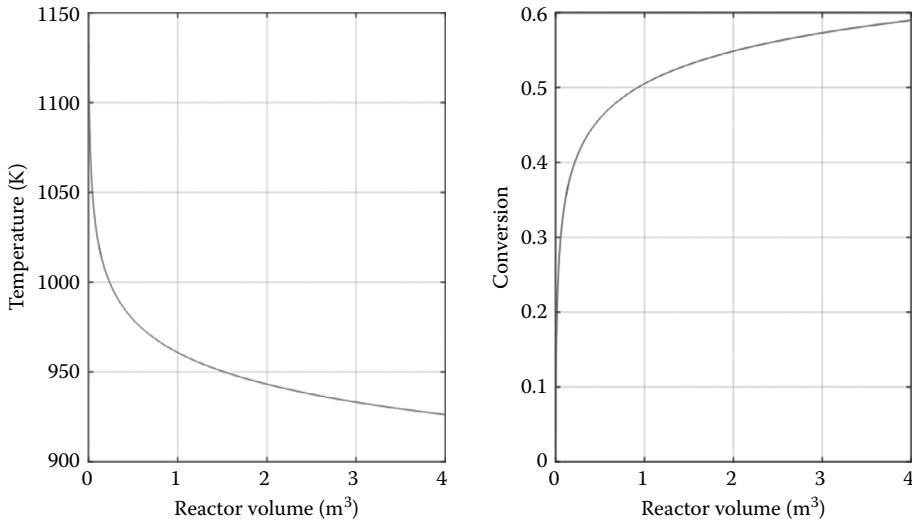


FIGURE 6.11 Temperature and conversion versus reactor volume.

```

>> xc = (X0(1) - X(:,1))/X0(1);
>> subplot(1, 2), plot(V,X(:,4)), xlabel('Reactor volume(m^3)')
>> ylabel('Temperature(K)'), grid
>> subplot(1, 2), plot(V,xc), xlabel('Reactor volume(m^3)'),
>> ylabel('Conversion'), grid

```

3. For acetone feed rates of $F_{A0} = 10, 20, 30, 35, 38.3$ mol/sec, the nitrogen flow rate is given by $F_{N_2} = 38.3 - F_{A0}$. The script *usead* calls the function *adfun* and calculates final conversions and temperatures using the built-in function *ode45*.

```

% usead.m: calculates final conversions and temperatures
P = [1.6:0.2:5]*101.325; Vspan = [0 4]; FA0 = [10 20 30 35 38.3];
FN2 = 38.3-FA0; nP = length(P); nF = length(FA0);
xc = zeros(nF,nP); T = zeros(nF,nP);
for i = 1:nF
    for j = 1:nP
        X0 = [FA0(i) 0 0 1035]; pf = [P(j) FN2(i)];
        [V X] = ode45(@adfun, Vspan, X0, [], pf);
        xc(i,j) = (X0(1) - X(end,1))/X0(1);
        T(i,j) = X(end,4);
    end
end
P = P/101.325;
figure(1)
plot(P,T(1,:),'o',P,T(2,:),'*',P,T(3,:),'x',P,T(4,:),'d',P,T(5,:),'v')
xlabel('P(atm)'), ylabel('T(K)'), grid
legend('F_A0=10','F_A0=20','F_A0=30','F_A0=35','F_A0=38.3')
figure(2)

```

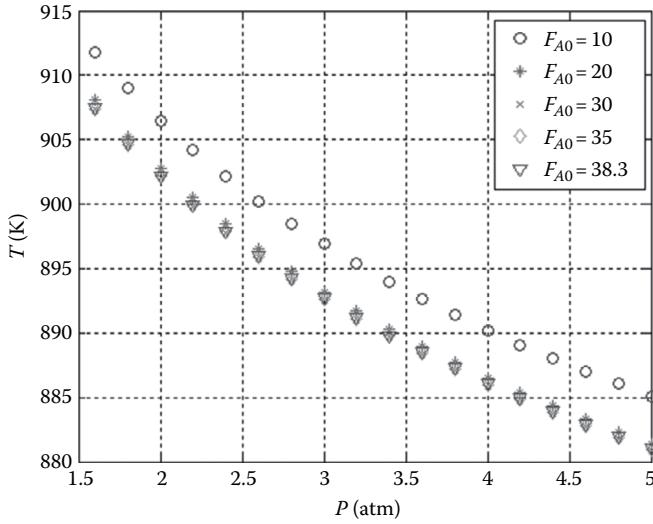


FIGURE 6.12 Temperature versus pressure and acetone flow rate.

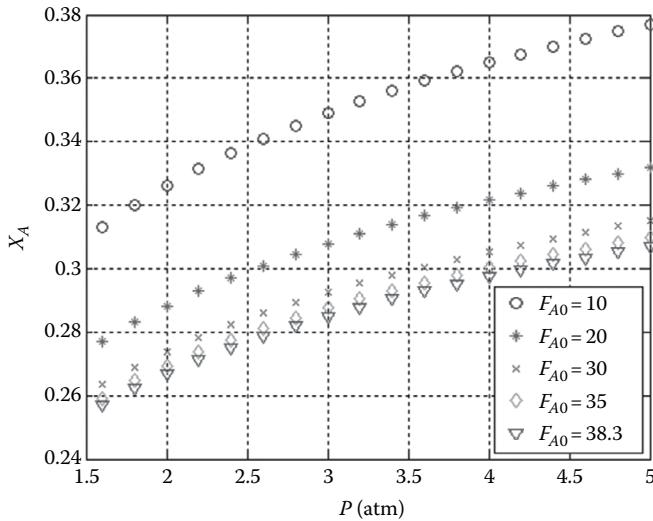


FIGURE 6.13 Final conversion versus pressure and acetone flow rate.

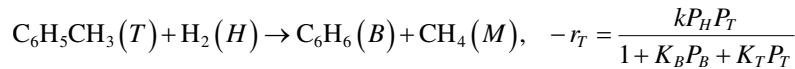
```
plot(P, xc(1, :), 'o', P, xc(2, :), '*', P, xc(3, :), 'x', P, xc(4, :), 'd', P, xc(5, :), 'v')
xlabel('P(atm)'), ylabel('x_A'), grid
legend('F_A0=10', 'F_A0=20', 'F_A0=30', 'F_A0=35', 'F_A0=38.3')
```

The script *usead* produces the plots shown in Figures 6.12 and 6.13:

```
>> usead
```

6.5 CATALYTIC REACTORS

Steps in a catalytic reaction consist of diffusion from the bulk to the external surface of the catalyst, surface reaction, and desorption. For the hydrodemethylation reaction in which toluene (*T*) reacts with hydrogen (*H*) to produce benzene (*B*) and methane (*M*), the reaction rate can be expressed by



where $P_i (i = T, H, B, M)$ is the partial pressure of each species and k, K_B, K_T are parameters.

Consider an isothermal gas-phase reaction being carried out in a packed bed catalytic reactor. The rate of change of conversion X with respect to the catalyst weight W can be expressed by

$$\frac{dX}{dW} = \frac{-r_A}{F_{A0}}$$

and the concentration of key component (A) is given by

$$C_A = C_{A0} \left(\frac{1-X}{1+\epsilon X} \right) \frac{P}{P_0} = C_{A0} \left(\frac{1-X}{1+\epsilon X} \right) y$$

where $y = \frac{P}{P_0}$, $\epsilon = y_{A0}\delta = \frac{F_{A0}}{F_{T0}}\delta$, and δ is the difference in stoichiometric ratio. When there is no change in the number of moles, $\epsilon = 0$. The pressure drop in a packed bed reactor is given by¹⁷

$$\frac{dy}{dW} = -\frac{\alpha}{2} \left(\frac{1+\epsilon X}{y} \right) \frac{T}{T_0}$$

where α is the pressure drop parameter. The pressure drop along the reactor length z is expressed by

$$\frac{dP}{dz} = -\beta_0 \frac{P_0}{P} \left(\frac{T}{T_0} \right) \frac{F_T}{F_{T0}}$$

where

β_0 is a constant related to a packed bed

P_0 and T_0 are the initial pressure and temperature

F is the molar flow rate

The ratio of volumetric flow rates is given by

$$f = \frac{v}{v_0} = \frac{1+\epsilon X}{y}$$

Example 6.14: Estimation of Catalytic Reaction Parameters¹⁸

Table 6.4 shows reaction rates and partial pressures for the hydrodemethylation reaction in which toluene (T) reacts with hydrogen (H) to produce benzene (B) and methane (M). The reaction rate can be expressed by

$$-r_T = \frac{kP_H P_T}{1 + K_B P_B + K_T P_T}$$

Use the regression method along with the data in **Table 6.4** to find the best estimates of the rate law parameters k, K_B , and K_T .

TABLE 6.4
Reaction Rate and Partial Pressure

$-r_T \times 10^{10}$	Partial Pressure (atm)			
	Toluene (P_T)	Hydrogen (P_H)	Methane (P_M)	Benzene (P_B)
71.0	1	1	1	0
71.3	1	1	4	0
41.6	1	1	0	1
19.7	1	1	0	4
42.0	1	1	1	1
17.1	1	1	0	5
71.8	1	1	0	0
142.0	1	2	0	0
284.0	1	4	0	0
47.0	0.5	1	0	0
71.3	1	1	0	0
117.0	5	1	0	0
127.0	10	1	0	0
131.0	15	1	0	0
133.0	20	1	0	0
41.8	1	1	1	1

Solution

Rearrangement of the given rate equation gives

$$-\frac{P_H P_T}{r_T} = \frac{1}{k} + \left(\frac{K_B}{k} \right) P_B + \left(\frac{K_T}{k} \right) P_T$$

We can use the linear regression method to estimate parameters. The script *pbrpar* employs the linear regression method to estimate the parameters and generates a plot of both measured and estimated reaction rates.

```
% pbrpar.m
r = 1e-10*[71.0 71.3 41.6 19.7 42.0 17.1 71.8 142.0 284.0 47.0 71.3 ...
117.0 127.0 131.0 133.0 41.8];
Pt = [1 1 1 1 1 1 1 1 0.5 1 5 10 15 20 1];
Ph = [1 1 1 1 1 1 2 4 1 1 1 1 1 1 1];
Pm = [1 4 0 0 1 0 0 0 0 0 0 0 0 0 0 1];
Pb = [0 0 1 4 1 5 0 0 0 0 0 0 0 0 0 1];
n = length(r); A = [ones(n,1) Pb' Pt'];
b = Ph.*Pt./r;
x = inv(A'*A)*A'*b';
k = 1/x(1), Kb = x(2)*k, Kt = x(3)*k
rc = k*Ph.*Pt./(1 + Kb*Pb + Kt*Pt); nc = 1:n;
plot(nc,r*1e10,'o',nc,rc*1e10,'*'), legend('Data','Estimated')
xlabel('Run #'), ylabel('-10^{10}r')
```

The script *pbrpar* calculates the rate law parameters k , K_B , and K_T and the plot is shown in Figure 6.14:

```
>> pbrpar
k = 1.40469e-08, Kb = 1, Kt = 1.00583
```

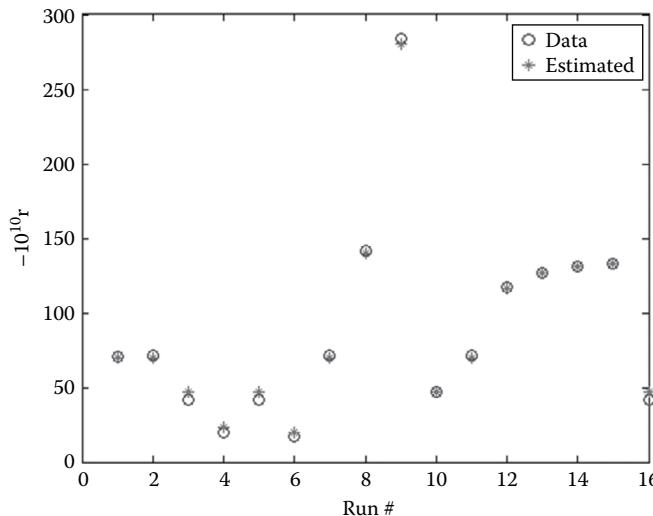
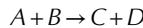


FIGURE 6.14 Comparison of reaction rates (o: data, *: estimated).

Example 6.15: Gas-Phase Reaction in a Packed Bed Reactor¹⁹

The irreversible gas-phase catalytic reaction



is to be carried out in a packed bed reactor with four different catalysts (Catalyst 1, Catalyst 2, Catalyst 3, Catalyst 4). For each catalyst, the rate expression has a different form:

$$\text{Catalyst 1: } -r_{A1} = \frac{kC_A C_B}{1 + K_A C_A}$$

$$\text{Catalyst 2: } -r_{A2} = \frac{kC_A C_B}{1 + K_A C_A + K_C C_C}$$

$$\text{Catalyst 3: } -r_{A3} = \frac{kC_A C_B}{(1 + K_A C_A + K_B C_B)^2}$$

$$\text{Catalyst 4: } -r_{A4} = \frac{kC_A C_B}{(1 + K_A C_A + K_B C_B + K_C C_C)^2}$$

The initial concentrations of the reactants are $C_{A0} = C_{B0} = 1.0 \text{ gmol/dm}^3$ at the reactor inlet, and the molar feed flow rate of A is $F_{A0} = 1.5 \text{ gmol/min}$. There is a total of $W_{\max} = 2 \text{ kg}$ of each catalyst used in the reactor. The reaction rate constant is $k = 10 \text{ dm}^6/(\text{kg} \cdot \text{min})$, and the various catalytic parameters K_A , K_B , and K_C are given by $K_A = 1 \text{ dm}^3/\text{gmol}$, $K_B = 2 \text{ dm}^3/\text{gmol}$, and $K_C = 20 \text{ dm}^3/\text{gmol}$, respectively. Calculate and plot the conversion X versus the catalyst weight W for each of the catalytic rate expressions when the reactor operation is at a constant pressure and $\alpha = 0.4$.

Solution

The mass balance for each catalyst yields $\frac{dX}{dW} = \frac{-r_{Ai}}{F_{A0}}$ ($i = 1, 2, 3, 4$). Since there is no change in the number of moles, $\epsilon = 0$, and we have $C_A = C_B = C_{A0}(1 - X)$, $C_C = C_D = C_{A0}X$. The MATLAB® function *pbrmf* defines the differential equations.

```
function dxdw = pbrmf(w, x, k, fa0, ca0, ka, kb, kc)
kfc = k*ca0^2/fa0;
```

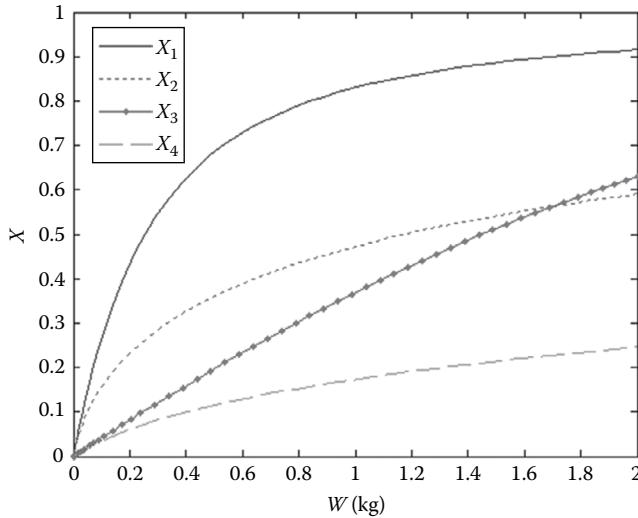


FIGURE 6.15 Conversion versus catalyst weight.

```

dxdw = [kfc*(1-x(1))^2/(1+ka*ca0*(1-x(1))) ;
        kfc*(1-x(2))^2/(1+ka*ca0*(1-x(2))+kc*ca0*x(2)) ;
        kfc*(1-x(3))^2/(1+ka*ca0*(1-x(3))+kb*ca0*(1-x(3)))^2 ;
        kfc*(1-x(4))^2/(1+ka*ca0*(1-x(4))+kb*ca0*(1-x(4))+kc*ca0*x(4))^2];
end

```

The built-in function *ode45* can be used to integrate the differential equations. The following commands generate the plot of the conversion versus the catalyst weight for each of the catalytic rate expressions (see [Figure 6.15](#)).

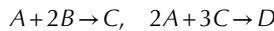
```

>> ca0=1; fa0=1.5; k=10; ka=1; kb=2; kc=20; wf=2; wint=[0 wf]; x0=[0 0 0 0];
>> [w x] = ode45(@pbrmf,wint,x0,[],k,fa0,ca0,ka,kb,kc);
>> plot(w,x(:,1),w,x(:,2),':',w,x(:,3),'.-',w,x(:,4), '--'), xlabel('W(kg)'), 
ylabel('X'))
>> legend('X_1','X_2','X_3','X_4')

```

Example 6.16: Complex Reactions in a Packed Bed Reactor²⁰

The following complex gas-phase reactions



take place isothermally in a packed bed reactor. These reactions follow elementary rate laws, and the reaction rate for each species can be written as

$$r_A = \frac{dF_A}{dW} = -k_a C_A C_B^2 - \frac{2}{3} k_c C_A^2 C_C^3$$

$$r_B = \frac{dF_B}{dW} = -2k_a C_A C_B^2$$

$$r_C = \frac{dF_C}{dW} = k_a C_A C_B^2 - k_c C_A^2 C_C^3$$

$$r_D = \frac{dF_D}{dW} = \frac{1}{3} k_c C_A^2 C_C^3$$

where

W is the catalyst weight (kg)

F_i is the molar flow rate of species i

The concentration of each species can be expressed by

$$C_i = C_{T0} \left(\frac{F_i}{F_T} \right) y \quad (i = A, B, C, D)$$

where $F_T = F_A + F_B + F_C + F_D$ and the rate of change of y with respect to W is given by

$$\frac{dy}{dW} = -\frac{\alpha}{2y} \left(\frac{F_T}{F_{T0}} \right), \quad y(0) = 1$$

The selectivity is defined by $S_{C/D} = \frac{F_C}{F_D}$. Plot the molar flow rate of each species and the selectivity

as a function of catalyst weight, W ($0 \leq W \leq 1000$ (kg)).

Data: $C_{T0} = 0.2$ mol/dm³, $\alpha = 0.0019$ kg⁻¹, $k_a = 100$ (dm³/mol)²/min/kg_{cat}, $k_c = 1500$ dm¹⁵/mol⁴/min/kg_{cat}, $F_{T0} = 20$ mol/min ($F_{A0} = F_{B0} = 10$ mol/min)

Solution

The MATLAB® function *pbrmult* defines the differential equations. In the function, the dependent variable vector x is defined as $x^T = [F_A \ F_B \ F_C \ F_D \ y]$.

```
function frx = pbrmult(W, x, ka, kc, Ct0, Ft0, alpha)
% x(1)=Fa, x(2)=Fb, x(3)=Fc, x(4)=Fd, x(5)=y
Ft = x(1) + x(2) + x(3) + x(4);
for i = 1:4
    C(i) = Ct0*x(i)*x(5)/Ft;
end
frx = [-ka*C(1)*C(2)^2 - 2*kc*C(1)^2*C(3)^3/3;
        -2*ka*C(1)*C(2)^2;
        ka*C(1)*C(2)^2 - kc*C(1)^2*C(3)^3;
        kc*C(1)^2*C(3)^3/3;
        - alpha*Ft/(2*x(5)*Ft0)];
end
```

The script *usepbrmult* calls the function *pbrmult*, employs the built-in function *ode45* to solve the differential equations, and generates plots of molar flow rates and the selectivity as a function of catalyst weight. To avoid division by zero, the value of the selectivity is set to zero when F_D is very small.

```
% usepbrmult.m
clear all;
ka = 100; kc = 1500; Ct0 = 0.2; Ft0 = 20; alpha = 0.0019;
x0 = [10 10 0 0 1]; Wv = [0 1000];
[W x] = ode45(@pbrmult,Wv,x0,[],ka,kc,Ct0,Ft0,alpha);
Fa = x(:,1); Fb = x(:,2); Fc = x(:,3); Fd = x(:,4); y = x(:,5);
n = length(W); Scd = zeros(1,n);
for i = 1:n
    if Fd(i) <= 1e-4
        Scd(i) = 0;
    else
        Scd(i) = Fc(i)/Fd(i);
    end
end
```

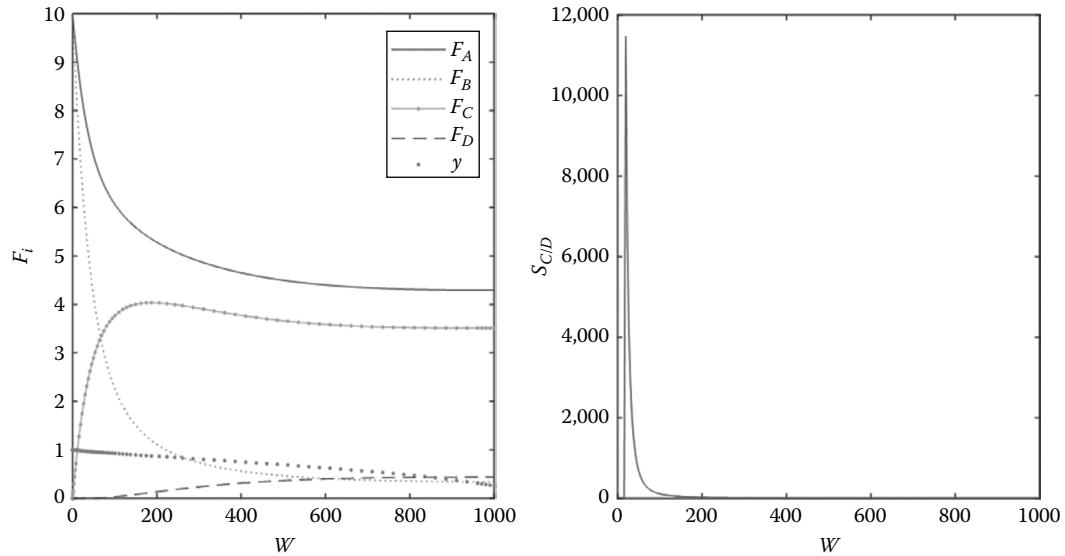


FIGURE 6.16 Molar flow rate and selectivity profiles.

```

subplot(1, 2), plot(W,Fa,W,Fb,':',W,Fc,'.-',W,Fd,'--',W,Y,'.')
xlabel('W'), ylabel('F_i'), legend('F_A','F_B','F_C','F_D','Y')
subplot(1, 2), plot(W,Scd), xlabel('W'), ylabel('S_{C/D}')
fprintf('Final molar flow rate of each species: \n');
fprintf(' Faf=%g, Fbf=%g, Fcf=%g\n',Fa(end),Fb(end),Fc(end),Fd(end));
fprintf('Final value of y: yf = %g\n',y(end));
fprintf('Selectivity: Scdf = %g\n',Scd(end));

```

The script *usepbrmult* produces the following outputs and generates Figure 6.16:

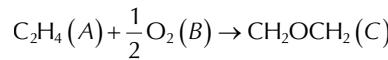
```

>> usepbrmult
Final molar flow rate of each species:
Faf=4.29365, Fbf=0.340968, Fcf=3.51426, Fdf=0.438418
Final value of y: yf = 0.257788
Selectivity: Scdf = 8.01579

```

Example 6.17: Conversion in a Packed Bed Reactor²¹

Ethylene oxide (C) can be produced by the vapor-phase catalytic oxidation of ethylene (A) with air (B):



$$-r_A = kP_A^{1/3}P_B^{2/3} = kP_{A0}\left(\frac{1}{2}\right)^{2/3}\left(\frac{1-X}{1+\epsilon X}\right)y = k'\left(\frac{1-X}{1+\epsilon X}\right)y$$

The ratio of the volumetric flow rate, f , is given by

$$f = \frac{V}{V_0} = \frac{1+\epsilon X}{y}$$

Calculate the catalyst weight necessary to achieve 60% conversion and plot X , y , f , and the reaction rate as a function of catalyst weight.

Data: $k = 0.00392 \text{ mol}/(\text{atm} \cdot \text{kg}_{\text{cat}} \cdot \text{sec})$, $F_{A0} = 0.1362 \text{ mol/sec}$, $F_{B0} = 0.068 \text{ mol/sec}$, $P_0 = 10 \text{ atm}$, $\alpha = 0.0367 \text{ kg}^{-1}$.

Solution

Substitution of reaction rate equation into the relation

$$\frac{dX}{dW} = \frac{-r_A}{F_{A0}}$$

gives

$$\frac{dX}{dW} = \frac{-r'_A}{F_{A0}} = \frac{k'}{F_{A0}} \left(\frac{1-X}{1+\epsilon X} \right) y, \quad X(0) = 0$$

Since temperature is constant, we have

$$\frac{dy}{dW} = -\frac{\alpha}{2} \left(\frac{1+\epsilon X}{y} \right) \frac{T}{T_0} = -\frac{\alpha}{2} \left(\frac{1+\epsilon X}{y} \right), \quad y(0) = 1$$

where

$$F_i = F_{B0} \left(\frac{79}{21} \right), \quad F_{T0} = F_{A0} + F_{B0} + F_i, \quad y_{A0} = \frac{F_{A0}}{F_{T0}}, \quad P_{A0} = y_{A0} P_0, \quad k' = k P_{A0} \left(\frac{1}{2} \right)^{2/3},$$

$$\delta = 1 - \frac{1}{2} - 1, \quad \epsilon = y_{A0} \delta.$$

Differential equations are integrated over the interval $W_{\text{span}} = [0 \quad W_f]$. The goal is to determine W_f at which the conversion $X = 0.6$ is achieved. This problem can be solved by using the bisection method. The initial search interval can be set as $15 \leq W_f \leq 25$. We guess the initial search interval and reduce the interval iteratively using the bisection method. The MATLAB® function *pbconv* defines differential equations. The script *usepbconv* calls the function *pbconv* and employs the built-in solver *ode45* to find solutions.

```
pbconv.m
function dz = pbconv(w, z, kp, Fa0, epsilon, alpha)
% x = z(1), y = z(2)
dz = [kp*(1-z(1))*z(2)/(Fa0*(1 + epsilon*z(1)));
      - alpha*(1 + epsilon*z(1))/(2*z(2))];
end

usepbconv.m
% usepbconv.m
clear all;
k = 0.00392; Fa0 = 0.1362; Fb0 = 0.068; P0 = 10; alpha = 0.0367;
Fi = Fb0*(79/21); Ft0 = Fa0 + Fb0 + Fi; ya0 = Fa0/Ft0;
Pa0 = ya0*P0; kp = k*Pa0*(0.5)^(2/3); delta = -0.5; epsilon = ya0*delta;
Wfa = 15; Wfb = 25; % initial guess of catalyst weight
z0 = [0 1]; Xf = 0.6;
while (abs(Wfa-Wfb) >= 1e-3)
    Wfm = (Wfa+Wfb)/2;
    [Wa Za] = ode45(@pbconv, [0 Wfa], z0, [], kp, Fa0, epsilon, alpha);
    [Wm Zm] = ode45(@pbconv, [0 Wfm], z0, [], kp, Fa0, epsilon, alpha);
    [Wb Zb] = ode45(@pbconv, [0 Wfb], z0, [], kp, Fa0, epsilon, alpha);
    Xa = Za(end,1); Xm = Zm(end,1); Xb = Zb(end,1);
    if (Xa-Xf)*(Xm-Xf) < 0
        Wfb = Wfm;
```

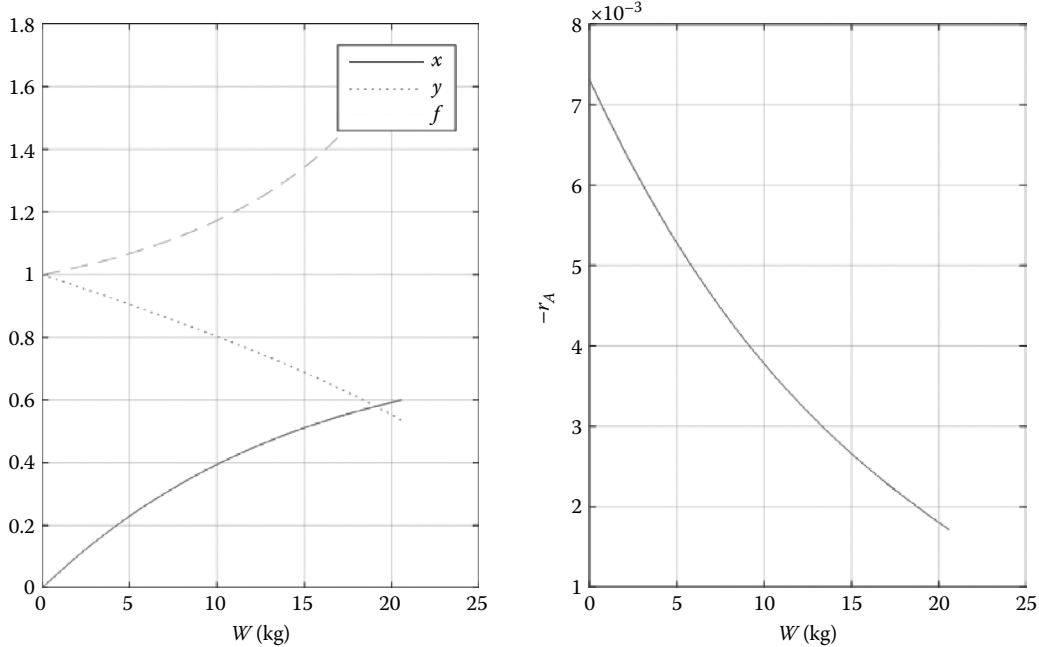


FIGURE 6.17 Conversion, flow ratio, and reaction rate versus catalyst weight.

```

else
Wfa = Wfm;
end
end
X = Zm(:,1); y = Zm(:,2); W = Wm;
fprintf('Catalyst weight = %g, Conversion = %g\n',Wfm, X(end));
fm = (1 + epsilon*X)./y; % ratio of the volumetric flow rate
rp = -kp*(1 - X).*y./(1 + epsilon*X); % reaction rate
subplot(1, 2), plot(Wm,X,Wm,y,':',Wm,fm,'--')
legend('X','y','f'), xlabel('W(kg)'), grid on
subplot(1, 2), plot(Wm,-rp), xlabel('W(kg)'), ylabel('-r_A'), grid on

```

The script *usepbconv* produces the desired results and plots X , y , f , and the reaction rate as a function of catalyst weight (see Figure 6.17).

```

>> usepbconv
Catalyst weight = 20.5634, Conversion = 0.599998

```

6.5.1 STRAIGHT-THROUGH TRANSPORT REACTOR

A catalytic gas-phase reaction



is carried out in a straight-through transport reactor where the catalyst bed is moving through the reactor. When the catalyst bed moves upward from the bottom of the reactor ($z = 0$), the mass balance yields

$$v_0 C_{A0} \frac{dx_A}{dz} = -r_A A_c, \quad x_A|_{z=0} = 0$$

where A_c is the cross-sectional area of the reactor. The reaction rate per unit volume of catalyst bed can be represented as the particular activity multiplied by the catalytic rate expression as

$$-r_A = \frac{akC_A}{1+K_AC_A}, \quad C_A = C_{A0}(1-x_A), \quad C_{B0} = C_{A0}x_A$$

where a denotes the activity of the catalyst. If the deactivation of the catalyst is negligible, the mass balance yields

$$\frac{dx_A}{dz} = \frac{akC_A}{(1+K_AC_A)u}, \quad u = \frac{v_0}{A_c}, \quad x_A|_{z=0} = 0$$

where u is the moving velocity of the catalyst bed.

When the deactivation of the catalyst is significant, three types of catalyst deactivation may be examined: coking, sintering, and poisoning. Let a_c , a_s , and a_p be activities of catalyst when deactivation is caused by coking, sintering, and poisoning, respectively. These activities are given by²²

$$a_c = \frac{1}{1+A'\sqrt{\frac{z}{u}}}, \quad \frac{da_s}{dz} = -\frac{k_{ds}a_s^2}{u}, \quad \frac{da_p}{dz} = -\frac{k_{dp}a_pC_B}{u}$$

Example 6.18: Cracking of Gas Oil²³

The rate of the gas-phase cracking reaction of a gas oil (A)



can be represented by

$$-r_A = \frac{akC_A}{1+K_AC_A}, \quad C_A = C_{A0}(1-x_A), \quad C_{B0} = C_{A0}x_A$$

The catalyst particles are assumed to move upward with the mean gas velocity given by $u = 8 \text{ m/sec}$. The reaction is to be carried out at 750°F under constant temperature and pressure. The volume change with reaction, pressure drop, and temperature variation may be neglected.

Plot the conversion of A (X_A) and the catalyst activity versus the reactor length z for the three types of catalyst deactivation (coking, sintering, and poisoning). The height of the reactor is $z_f = 6 \text{ m}$, and the initial activity of the catalyst is assumed to be 1.

Data: $k = 30 \text{ sec}^{-1}$, $K_A = 5 \text{ m}^3/\text{kgmol}$, $C_{A0} = 0.2 \text{ kgmol/m}^3$, $A' = 12 \text{ sec}^{-1/2}$, $k_{ds} = 17.5 \text{ sec}^{-1}$, $k_{dp} = 140 \text{ dm}^3/(\text{mol} \cdot \text{sec})$

Solution

Let $X_A(i = 1, 2, 3)$ be the conversion for each deactivation type (1: coking, 2: sintering, 3: poisoning) and let $a_i(i = 1, 2, 3)$ be the corresponding catalyst activities. If we define the variable vector y as $y(i) = x_A(i = 1, 2, 3)$ and $y(i) = a_{i-2}(i = 4, 5)$, we can have the following differential equations:

$$\frac{dy(1)}{dz} = \frac{a_1 k C_{A0} (1-y(1))}{(1+K_A C_{A0} (1-y(1)))u}, \quad \frac{dy(2)}{dz} = \frac{y(4) k C_{A0} (1-y(2))}{(1+K_A C_{A0} (1-y(2)))u}, \quad a_1 = \frac{1}{1+A'\sqrt{\frac{z}{u}}},$$

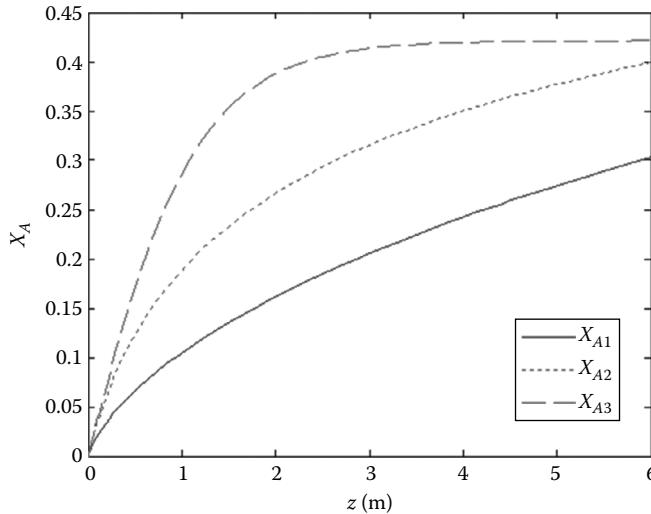


FIGURE 6.18 Conversion versus reactor height.

$$\frac{dy(3)}{dz} = \frac{y(5)kC_{A0}(1-y(3))}{(1+K_A C_{A0}(1-y(3)))u}, \quad \frac{dy(4)}{dz} = -\frac{k_{ds}y(4)^2}{u}, \quad \frac{dy(5)}{dz} = -\frac{k_{dp}y(5)C_{A0}y(3)}{u}$$

The script *mcatb* defines these differential equations and uses the built-in solver *ode45* to find solutions.

```
% mcatb.m
% y1=Xa1, y2=Xa2, y3=Xa3, y4=a2, y5=a3
clear all;
u=8; Ap=12; k=30; Ka=5; Ca0=0.2; kds=17.5; kdp=140;
zspan = [0 6]; y0 = [0 0 0 1 1];
dydz = @(z,y) [(1/(1+Ap*sqrt(z/u)))*k*Ca0*(1-y(1))/(1+Ka*Ca0*(1-y(1)))/u;
                y(4)*k*Ca0*(1-y(2))/(1+Ka*Ca0*(1-y(2)))/u;
                y(5)*k*Ca0*(1-y(3))/(1+Ka*Ca0*(1-y(3)))/u;
                -kds*y(4)^2/u;
                -kdp*Ca0*y(3)*y(5)/u];
[z,y] = ode45(dydz, zspan, y0);
figure(1), plot(z,y(:,1),'-',z,y(:,2),':',z,y(:,3), '--');
xlabel('z(m)'), ylabel('X_A'), legend('X_{A1}', 'X_{A2}', 'X_{A3}');
figure(2), plot(z,y(:,4),'-',z,y(:,5),':');
xlabel('z(m)'), ylabel('a'), legend('a_s', 'a_p');
```

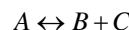
The script *mcatb* produces the curves shown in Figures 6.18 and 6.19.

```
>> mcatb
```

6.6 MEMBRANE REACTORS

The membrane can either provide a barrier to certain components while being permeable to others, prevent certain components such as particulates from contacting the catalyst, or contain reactive sites and be a catalyst in itself. Figure 6.20 shows a schematic of an inert membrane reactor with catalyst pellets on the feed side.²⁴

Consider a reversible reaction



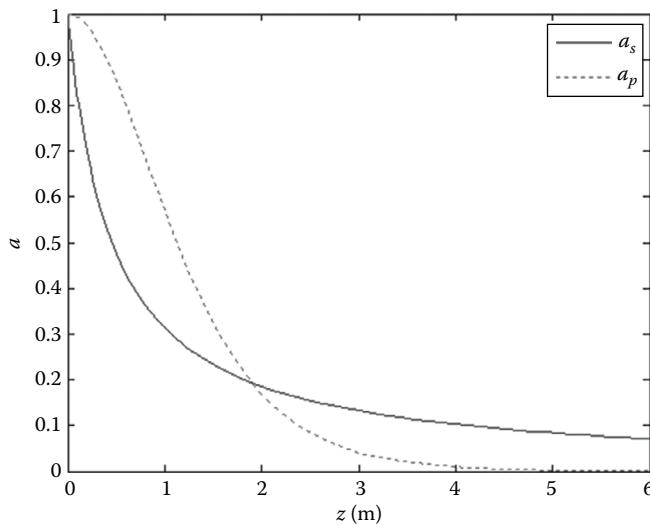


FIGURE 6.19 Catalyst activity versus reactor height.

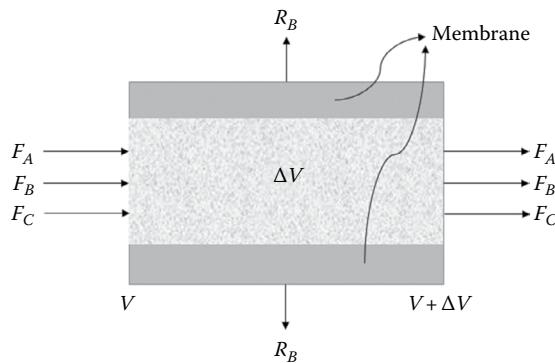


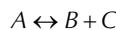
FIGURE 6.20 Schematic of membrane reactor with catalyst pellets.

taking place on the catalyst side of a membrane reactor with catalyst. The mass balance for each component gives

$$\frac{dF_A}{dV} = r_A, \quad \frac{dF_B}{dV} = r_B - R_B, \quad \frac{dF_C}{dV} = r_C, \quad R_B = k_c C_B$$

Example 6.19: Membrane Reactor²⁵

A reversible reaction



takes place on the catalyst side of a membrane reactor with catalyst pellets. The reaction rate and the rate of diffusion of the product *B* out of the reactor, R_B , are given by

$$-r_A = k \left(C_A - \frac{C_B C_C}{K_C} \right), \quad R_B = k_c C_B$$

Data: $k = 0.7 \text{ min}^{-1}$, $K_C = 0.05 \text{ mol/dm}^3$, $k_c = 0.2 \text{ min}^{-1}$, $F_{A0} = 10 \text{ mol/min}$, $F_{B0} = F_{C0} = 0$, $P = 830.6 \text{ kPa}$, $T = 500 \text{ K}$, $R = 8.314 \text{ kPa} \cdot \text{dm}^3/(\text{mol} \cdot \text{K})$

1. Plot the molar flow rates of each species as a function of reactor volume V ($0 \leq V \leq 500 \text{ (dm}^3\text{)}$).
2. Calculate the conversion of A at $V = 400 \text{ dm}^3$.

Solution

Relative reaction rates yield

$$\frac{r_A}{-1} = \frac{r_B}{1} = \frac{r_C}{1} \Rightarrow r_B = -r_A, r_C = -r_A$$

The concentration of each species is given by

$$C_i = C_{T0} \left(\frac{F_i}{F_T} \right) \quad (i = A, B, C), \quad C_{T0} = \frac{P}{RT}$$

The mass balance on each component gives

$$\frac{dF_A}{dV} = r_A, \quad \frac{dF_B}{dV} = r_B - R_B = -r_A - k_c C_{T0} \left(\frac{F_B}{F_T} \right), \quad \frac{dF_C}{dV} = r_C = -r_A$$

The combination of the reaction rate equation and concentration equations yields

$$-r_A = k \left(C_A - \frac{C_B C_C}{K_C} \right) = k C_{T0} \left[\frac{F_A}{F_T} - \frac{C_{T0}}{K_C} \left(\frac{F_B}{F_T} \right) \left(\frac{F_C}{F_T} \right) \right], \quad F_T = F_A + F_B + F_C$$

The function *membrx* defines differential equations to be solved.

```
function dxdv = membrx(v, x, k, kc, Kc, Ct0)
% x(1) = Fa, x(2) = Fb, x(3) = Fc
Ft = sum(x);
rA = -k*Ct0*(x(1)/Ft - Ct0*x(2)*x(3)/(Kc*Ft^2));
dxdv = [rA;
         -rA - kc*Ct0*x(2)/Ft;
         -rA];
end
```

The script *usemembrx* calls the function *membrx* and employs the built-in function *ode45* to solve the system of differential equations.

```
% usemembrx.m
Fa0 = 10; k = 0.7; kc = 0.2; Kc = 0.05; R = 8.314; P = 830.6; T = 500;
Ct0 = P/(R*T); Vf = 500; Vc = 400; x0 = [Fa0 0 0];
[V x] = ode45(@membrx, [0 Vf], x0, [], k, kc, Kc, Ct0);
Fa = x(:,1); Fb = x(:,2); Fc = x(:,3);
fprintf('Final values: Fa = %g, Fb = %g, Fc = %g\n', Fa(end), Fb(end),
Fc(end));
plot(V, Fa, V, Fb, ':', V, Fc, '--'), xlabel('V(dm^3)'), axis tight
ylabel('Molar flow rate (mol/min)'), legend('F_A', 'F_B', 'F_C')
for i = 1:length(V)
    if V(i) >= Vc
        iV = i; break;
    end
end
```

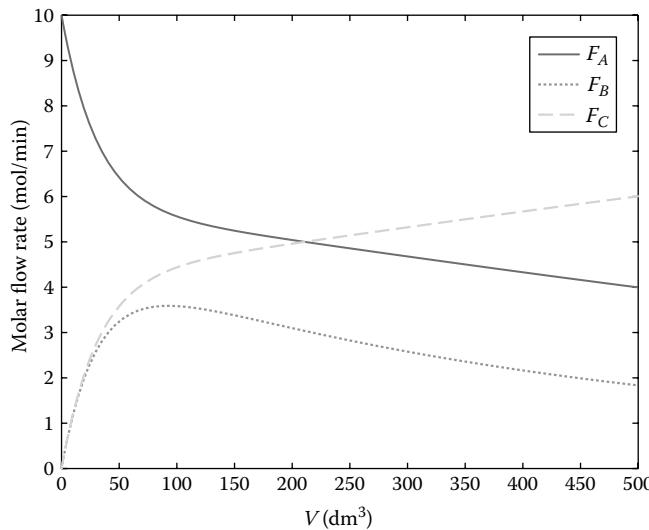


FIGURE 6.21 Molar flow rates versus reactor volume.

```

Fc = Fa(iV) + (Fa(iV+1)-Fa(iV))*(Vc - V(iV))/(V(iV+1)-V(iV));
Xa = (Fa0 - Fc)/Fa0;
fprintf('At V = %g, Fa = %g and the conversion of A = %g\n', Vc, Fc, Xa);

```

The script *usemembrx* produces the following results and Figure 6.21:

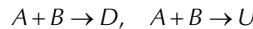
```

>> usemembrx
Final values: Fa = 3.99507, Fb = 1.83459, Fc = 6.00493
At V = 400, Fa = 4.32946 and the conversion of A = 0.567054

```

Example 6.20: Multiple Reactions in a Membrane Reactor²⁶

The gas-phase reactions



take place in a membrane reactor. The reaction rate for each component is given by

$$r_A = -k_1 C_A^2 C_B - k_2 C_A C_B^2, \quad r_B = -k_1 C_A^2 C_B - k_2 C_A C_B^2, \quad r_D = k_1 C_A^2 C_B, \quad r_U = k_2 C_A C_B^2$$

and the concentration of each species is given by

$$C_i = C_{T0} \frac{F_i}{F_T} \quad (i = A, B, D, U)$$

The component mass balance on each species yields

$$F_T = F_A + F_B + F_D + F_U, \quad \frac{dF_A}{dV} = r_A, \quad \frac{dF_B}{dV} = r_B + R_B, \quad \frac{dF_D}{dV} = r_D, \quad \frac{dF_U}{dV} = r_U$$

where $R_B = \frac{F_{B0}}{V_t}$ and V_t is the reactor volume (dm^3). The selectivity is defined by $S_{D/U} = \frac{F_D}{F_U}$.

Plot the molar flow rates and the overall selectivity as a function of reactor volume V ($0 \leq V \leq 50(\text{dm}^3)$). The molar flow rate of A entering the reactor is 4 mol/sec and that of B entering through the membrane, F_{B0} , is 4 mol/sec.

Data: $k_1 = 2 \text{ dm}^6/\text{mol}^2 \cdot \text{sec}$, $k_2 = 3 \text{ dm}^6/\text{mol}^2 \cdot \text{sec}$, $C_{T0} = 0.8 \text{ mol}/\text{dm}^3$, $V_t = 50 \text{ dm}^3$

Solution

The function *mbrmult* defines differential equations. In the function, the dependent variable vector x is defined by $x^T = [F_A \ F_B \ F_D \ F_U]$.

```
function frx = mbrmult(V,x,k1,k2,Ct0,Rb)
% x(1)=Fa, x(2)=Fb, x(3)=Fd, x(4)=Fu
C = Ct0*x/sum(x);
frx = [-k1*C(1)^2*C(2) - k2*C(1)*C(2)^2;
        -k1*C(1)^2*C(2) - k2*C(1)*C(2)^2 + Rb;
        k1*C(1)^2*C(2);
        k2*C(1)*C(2)^2];
end
```

The script *usembrmult* calls the function *mbrmult* and uses the built-in function *ode45* to find solutions. The initial value of the vector x is set to $x_0 = [4 \ 0 \ 0 \ 0]$ and that of F_U is set to zero. But, when the value of F_U is very small, the value of the overall selectivity is set to zero to avoid division by zero.

```
% usembrmult.m
clear all;
k1 = 2; k2 = 3; Ct0 = 0.8; Fb0 = 4; Vt = 50; Fai = 4;
Rb = Fb0/Vt; x0 = [Fai 0 0 0]; Vspan = [0 Vt];
[V x] = ode45(@mbrmult,Vspan,x0,[],k1,k2,Ct0,Rb);
Fa = x(:,1); Fb = x(:,2); Fd = x(:,3); Fu = x(:,4);
n = length(V); Sdu = zeros(1,n);
for i = 1:n
    if Fu(i) <= 1e-6
        Sdu(i) = 0;
    else
        Sdu(i) = Fd(i)/Fu(i);
    end
end
subplot(1, 2), plot(V,Fa,V,Fb,':',V,Fd,'.-',V,Fu,'--')
xlabel('V(dm^3)'), ylabel('F_i(mol/s)')
legend('F_A','F_B','F_D','F_U','Location','best')
subplot(1, 2), plot(V,Sdu), xlabel('V(dm^3)'), ylabel('S_{D/U}')
fprintf('Final molar flow rates: \n');
fprintf('Faf=%g, Fbf=%g, Fdf=%g, Fuf=%g\n',Fa(end),Fb(end),Fd(end),Fu(end));
fprintf('Overall selectivity: Sduf = %g\n',Sdu(end));
```

The script *usembrmult* produces the following outputs and Figure 6.22:

```
>> usembrmult
Final molar flow rates:
Faf=1.35139, Fbf=1.35139, Fdf=1.90998, Fuf=0.738632
Overall selectivity: Sduf = 2.58584
```

6.7 BIOREACTORS

While many models exist for the growth rate of new cells, the most commonly used expression is the Monod equation given by

$$r_g = \mu C_c$$

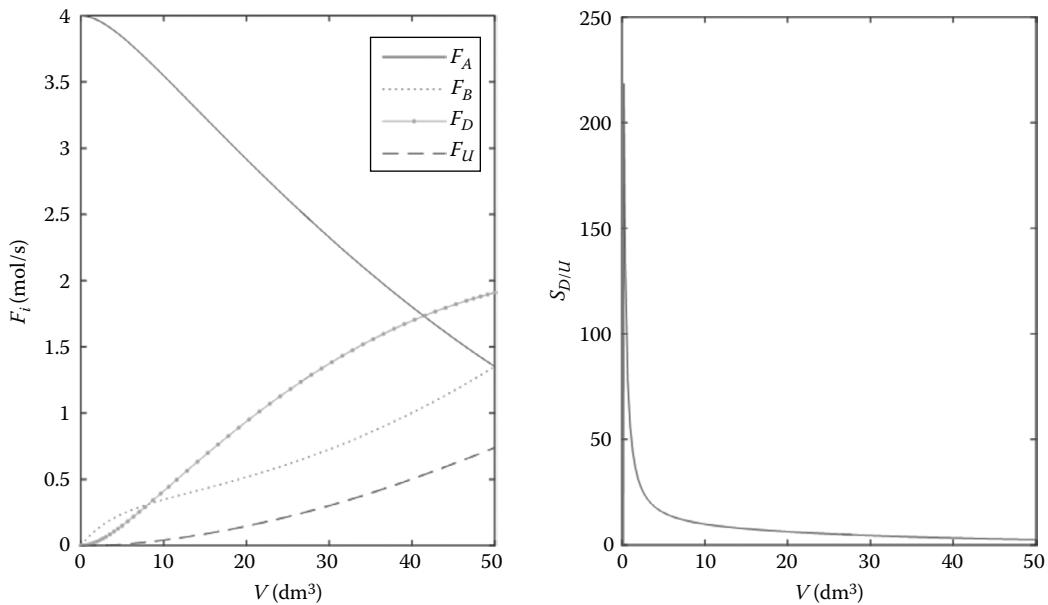


FIGURE 6.22 Molar flow rates and overall selectivity versus reactor volume.

where

r_g is the cell growth rate ($\text{g}/(\text{dm}^3 \cdot \text{sec})$)

C_c is the cell concentration (g/dm^3)

μ denotes the specific growth rate (sec^{-1}) given by

$$\mu = \mu_{max} \frac{C_s}{K_s + C_s}$$

where

μ_{max} is the maximum specific growth reaction rate (sec^{-1})

K_s is the Monod constant (g/dm^3)

C_s is the substrate concentration (g/dm^3)

A combination of these equations yields the Monod equation for bacterial cell growth rate

$$r_g = \frac{\mu_{max} C_s C_c}{K_s + C_s}$$

The yield coefficients are defined as

$$Y_{c/s} = -\frac{\Delta C_c}{\Delta C_s}, \quad Y_{p/c} = \frac{\Delta C_p}{\Delta C_c}, \quad Y_{p/s} = -\frac{\Delta C_p}{\Delta C_s}, \quad Y_{s/c} = \frac{1}{Y_{c/s}}, \quad Y_{c/p} = \frac{1}{Y_{p/c}}, \quad Y_{s/p} = \frac{1}{Y_{p/s}}$$

where C_p is the concentration of the product (g/dm^3) and the subscripts c , p , and s represent cell, product, and substrate, respectively.

For a batch system, the cell mass balance gives

$$\frac{dC_c}{dt} = r_g - r_d$$

where $r_g - r_d$ denotes the net rate of formation of living cell. The rate of substrate consumption, r_{sm} , is given by

$$r_{sm} = mC_c$$

The mass balance on the substrate yields²⁷

$$\frac{dC_s}{dt} = Y_{s/c}(-r_g) - mC_c$$

The rate of product formation, r_p , can be related to the net rate of substrate consumption, $-r_s$, through the following balance when $m = 0$:

$$V \frac{dC_p}{dt} = r_p V = Y_{p/s}(-r_s) V$$

Example 6.21: Bacteria Growth in a Batch Reactor²⁸

Glucose-to-ethanol fermentation is to be carried out in a batch reactor using an organism. The cell growth rate r_g , the cell consumption rate r_d , and the substrate consumption rate r_{sm} are given by

$$r_g = \mu_{max} \left(1 - \frac{C_p}{C_p^*} \right)^{0.52} \frac{C_c C_s}{K_s + C_s}, \quad r_d = k_d C_c, \quad r_{sm} = m C_c$$

respectively. Plot the concentrations of cells, substrate, and product and the rates r_g , r_d , and r_{sm} as functions of time.

Data: the initial cell concentration = 1 g/dm³, the substrate (glucose) concentration = 250 g/dm³, $C_p^* = 93$ g/dm³, $Y_{c/s} = 0.08$ g/g, $Y_{p/s} = 0.45$ g/g, $Y_p = 5.6$ g/g, $n = 0.52$, $\mu_{max} = 0.33$ hr⁻¹, $K_s = 1.7$ g/dm³, $k_d = 0.01$ hr⁻¹, $m = 0.03$ g/g/hr

Solution

Substitution of rate equations into the mass balance equations

$$\frac{dC_c}{dt} = r_g - r_d, \quad \frac{dC_s}{dt} = Y_{s/c}(-r_g) - r_{sm}, \quad \frac{dC_p}{dt} = Y_{p/s} r_g$$

gives the following relations:

$$\frac{dC_c}{dt} = \mu_{max} \left(1 - \frac{C_p}{C_p^*} \right)^{0.52} \frac{C_c C_s}{K_s + C_s} - k_d C_c, \quad \frac{dC_s}{dt} = -Y_{s/c} \mu_{max} \left(1 - \frac{C_p}{C_p^*} \right)^{0.52} \frac{C_c C_s}{K_s + C_s} - m C_c$$

The function *geferm* defines the system of differential equations. In the function, the dependent variable *C* is defined by $C^T = [C_c \quad C_s \quad C_p]$.

```
function dC = geferm(t, C, mumax, Cps, Ks, kd, m, Ysc, Ypc)
% C(1)=Cc, C(2)=Cs, C(3)=Cp
rd = kd*C(1); rsm = m*C(1);
rg = mumax*(1 - C(3)/Cps)^0.52 * C(1)*C(2)/(Ks + C(2));
dC = [rg - rd;
       -rg*Ysc - rsm;
       Ypc*rg];
end
```

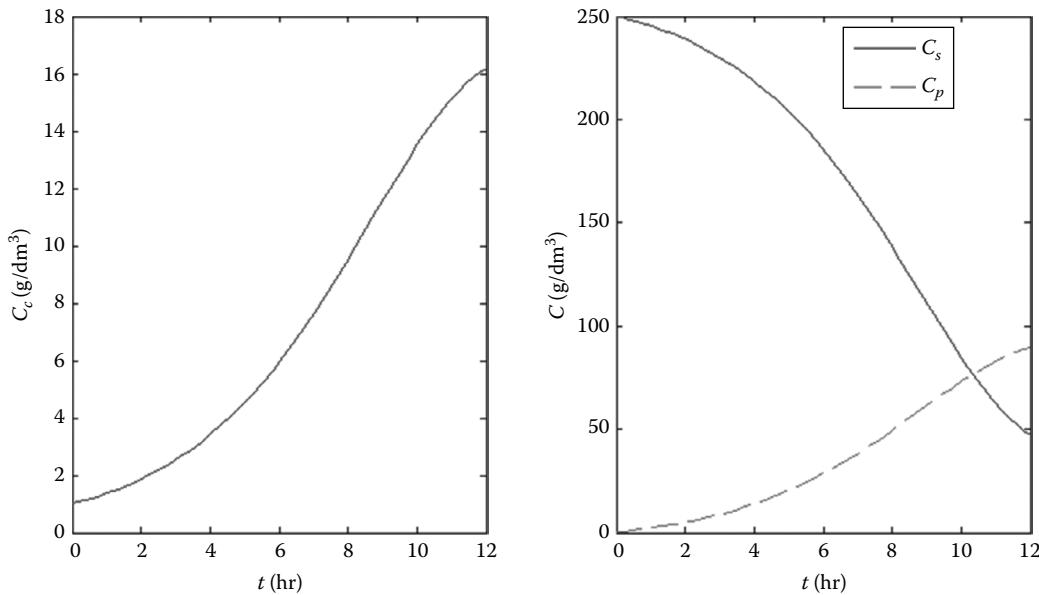


FIGURE 6.23 Concentrations as a function of time.

The script *usegeferm* calls the function *geferm* and employs the built-in function *ode45* to solve the system of differential equations. The initial value of C is set to $C_0 = [1 \ 250 \ 0]$.

```
% usegeferm.m
clear all;
mumax=0.33; Cps=93; Ks=1.7; kd=0.01; m=0.03; Ysc=1/0.08; Ypc=5.6;
C0 = [1 250 0]; tspan = [0 12];
[t C] = ode45(@geferm,tspan,C0,[],mumax,Cps,Ks,kd,m,Ysc,Ypc);
Cc = C(:,1); Cs = C(:,2); Cp = C(:,3);
rd = kd*Cc; rsm = m*Cc; rg = mumax*(1-Cp/Cps).^0.52 .* Cc.*Cs./(Ks+Cs);
figure(1)
subplot(1, 2), plot(t,Cc), xlabel('t(hr)'), ylabel('C_c(g/dm^3)')
subplot(1, 2), plot(t,Cs,t,Cp,'--'), xlabel('t(hr)'), ylabel('C(g/dm^3)')
legend('C_s','C_p','Location','best')
figure(2), plot(t,rg,t,rsm,'--',t,rd,'--')
xlabel('t(hr)'), ylabel('rates(g/dm^3/hr)')
legend('r_g','r_sm','r_d','Location','best')
fprintf('Final concentrations: Ccf=%g, Csf=%g, Cpf=%g\n',Cc(end),Cs(end),Cp(end));
fprintf('Final reaction rates: rgf=%g, rsmf=%g, rdf=%g\n',rg(end),rsm(end),rd(end));
```

The script *usegeferm* produces the following outputs and [Figures 6.23](#) and [6.24](#):

```
>> usegeferm
Final concentrations: Ccf=16.1841, Csf=46.9352, Cpf=89.8229
Final reaction rates: rgf=0.890418, rsmf=0.485522, rdf=0.161841
```

6.8 NONISOTHERMAL REACTIONS

6.8.1 ADIABATIC REACTION

The elementary gas-phase reversible reaction



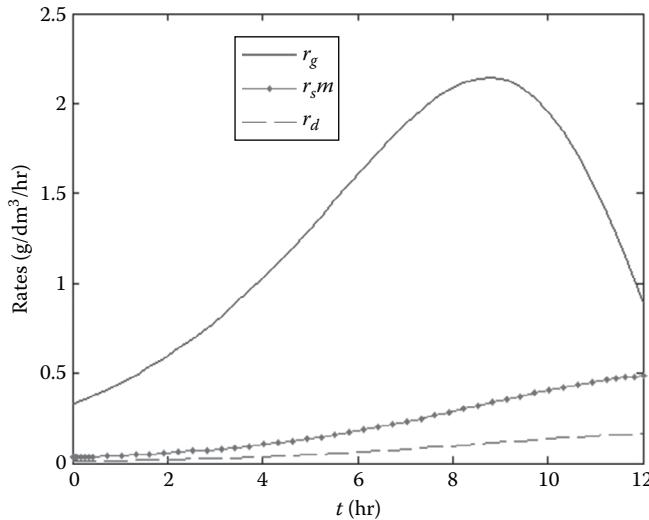


FIGURE 6.24 Reaction rates as a function of time.

is carried out in a plug-flow reactor in which pressure drop is neglected ($P = P_0$) and pure A enters the reactor. The mole balance gives

$$\frac{dX}{dV} = \frac{-r_A}{F_{A0}}$$

where F_{A0} is the molar flow rate of component A fed into the reactor. The reaction rate and rate constants can be expressed by

$$-r_A = k \left(C_A - \frac{C_B}{K_C} \right), \quad k = k_1(T_1) \exp \left[\frac{E}{R} \left(\frac{1}{T_1} - \frac{1}{T} \right) \right], \quad K_C = K_{C2}(T_2) \exp \left[\frac{\Delta H_{R_x}^\circ}{R} \left(\frac{1}{T_2} - \frac{1}{T} \right) \right]$$

where $k_1(T_1)$ and $K_{C2}(T_2)$ are values of k and K_C at T_1 and T_2 , respectively. For a gas-phase reaction, $\epsilon = 0$, and we have

$$C_A = C_{A0}(1-X) \frac{T_0}{T}, \quad C_B = C_{A0}X \frac{T_0}{T}$$

where T_0 is the temperature of the feed stream. The combination of these equations yields

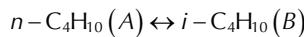
$$-r_A = k C_{A0} \left[\left(1 - X \right) - \frac{X}{K_C} \right] \frac{T_0}{T}$$

From the energy balance on the reactor, T can be expressed as a function of conversion as

$$T = \frac{X \left[-\Delta H_{R_x}^\circ (T_R) \right] + \sum \Theta_i C_{pi} T_0 + X \Delta C_p T_R}{\sum \Theta_i C_{pi} + X \Delta C_p}$$

Example 6.22: Adiabatic Liquid-Phase Isomerization of *n*-butane²⁹

n-Butane (C_4H_{10}) is to be isomerized to isobutane in a plug-flow reactor:



The reaction is an elementary reversible reaction to be carried out adiabatically in the liquid phase under high pressure using essentially trace amounts of a liquid catalyst. The feed enters at $T_0 = 330$ K. At $T_1 = 360$ K and $T_2 = 333$ K, it is known that $k_1(T_1) = 31.1 \text{ hr}^{-1}$ and $K_{C2}(T_2) = 3.03 \text{ hr}^{-1}$. A mixture of 90 mol % *n*-butane and 10 mol % *i*-pentane, which is considered an inert, is to be processed at 70% conversion. The molar flow rate of the mixture is 163 kmol/hr. Plot conversion X , equilibrium conversion X_e , temperature T , and reaction rate $-r_A$ down the length of the reactor.

Data: $\Delta H_{R_x}^\circ = -6,900 \text{ J/mol}$, activation energy $E = 65,700 \text{ J/mol}$, $C_{A0} = 9.3 \text{ kmol/m}^3$, $R = 8.314 \text{ J/(mol}\cdot\text{K)}$, $C_{p_{n-B}} = 141 \text{ J/(mol}\cdot\text{K)}$, $C_{p_{i-B}} = 141 \text{ J/(mol}\cdot\text{K)}$, $C_{p_{i-p}} = 161 \text{ J/(mol}\cdot\text{K)}$.

Solution

$$F_{A0} = 0.9F_0 = (0.9)(163) = 146.7 \text{ kmol/hr}, \quad T_0 = 330 \text{ K.}$$

$$\sum \Theta_i C_{pi} = C_{pA} + \Theta_i C_{pl} = \left(141 + \frac{0.1}{0.9} 161 \right) = 158.889 \text{ J/(mol}\cdot\text{K)}.$$

Since $\Delta C_p = C_{pB} - C_{pA} = 141 - 141 = 0$, we have

$$T = \frac{X[-\Delta H_{R_x}^\circ(T_R)] + \sum \Theta_i C_{pi} T_0}{\sum \Theta_i C_{pi}} = T_0 + \frac{X[-\Delta H_{R_x}^\circ(T_R)]}{\sum \Theta_i C_{pi}} = 330 + \frac{-(-6900)}{158.889} X = 330 + 43.4265 X$$

Substitution of concentrations $C_A = C_{A0}(1 - X)$ and $C_B = C_{A0}X$ to the reaction rate equation gives

$$-r_A = kC_{A0} \left[1 - \left(1 + \frac{1}{K_c} \right) X \right]$$

The change of conversion with respect to reactor volume is given by

$$\frac{dX}{dV} = \frac{-r_A}{F_{A0}}$$

At equilibrium, $-r_A = 0$, and the equilibrium conversion is given by

$$X_e = \frac{K_c}{1 + K_c}$$

The function *adbptr* defines the differential equation:

```
function dX = adbptr(V,X,Ca0,Fa0,T1,T2,k1,K2,E,R,dH)
T = 330 + 43.4265*X;
k = k1*exp(E*(1/T1 - 1/T)/R); Kc = K2*exp(dH*(1/T2 - 1/T)/R);
ra = -k*Ca0*(1 - (1 + 1/Kc)*X);
dX = -ra/Fa0;
end
```

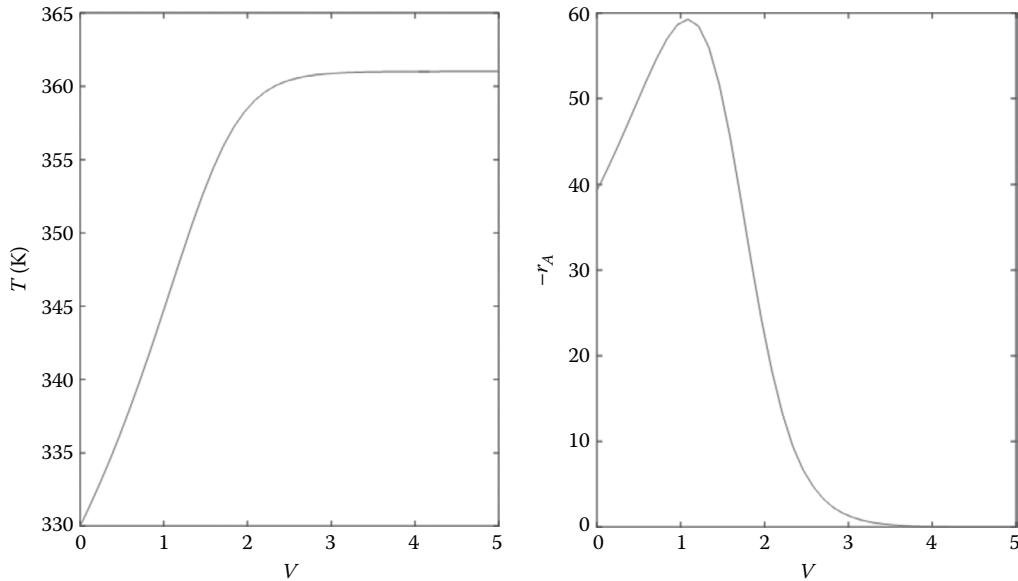


FIGURE 6.25 Temperature and reaction rate profiles.

The script *useadbpf* sets data and employs the built-in function *ode45* to solve the differential equation defined by the function *adbpf*.

```
% useadbpf.m
clear all;
Ca0 = 9.3; Fa0 = 146.7; T1 = 360; T2 = 333; k1 = 31.1; K2 = 3.03;
E = 65700; R = 8.314; dH = -6900; Vspan = [0 5]; X0 = 0;
[V X] = ode45(@adbpf,Vspan,X0,[],Ca0,Fa0,T1,T2,k1,K2,E,R,dH);
T = 330 + 43.4265*X;
k = k1*exp(E*(1/T1 - 1./T)/R); Kc = K2*exp(dH*(1/T2 - 1./T)/R);
ra = -k*Ca0.* (1 - (1 + 1./Kc).*X); Xe = Kc./(1+Kc);
figure(1)
subplot(1, 2), plot(V,T), xlabel('V'), ylabel('T(K)')
subplot(1, 2), plot(V,-ra), xlabel('V'), ylabel('-r_A')
figure(2)
plot(V,X,V,Xe,'--'), xlabel('V'), ylabel('X,X_e'), legend('X','Xe')
fprintf('Conversion (X) and equilibrium conversion (Xe): Xf = %g,
Xef = %g\n',X(end),Xe(end));
fprintf('Final temperature: Tf = %g\n',T(end));
fprintf('Final reaction rate: raf = %g\n',-ra(end));
```

The script *useadbpf* produces the following outputs and [Figures 6.25](#) and [6.26](#):

```
>> useadbpf
Conversion (X) and equilibrium conversion (Xe): Xf = 0.714056, Xef = 0.714063
Final temperature: Tf = 361.009
Final reaction rate: raf = 0.00312054
```

6.8.2 STEADY-STATE NONISOTHERMAL REACTIONS

Consider the reversible gas-phase reaction



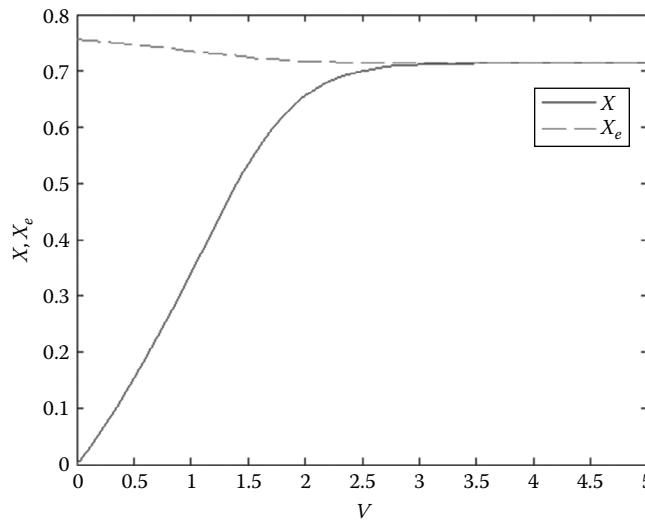


FIGURE 6.26 Conversion and equilibrium conversion profile.

to be carried out in a plug-flow reactor or a packed bed reactor in which pressure drop is neglected ($P = P_0$). The mole balance on each species yields

$$\frac{dX}{dV} = \frac{-r_A}{F_{A0}}, \quad \frac{dF_A}{dV} = r_A, \quad \frac{dF_B}{dV} = r_B, \quad \frac{dF_C}{dV} = r_C$$

where the reaction rates can be expressed as

$$-r_A = k_1 \left(C_A C_B - \frac{C_C^2}{K_C} \right), \quad r_B = r_A, \quad r_C = -2r_A$$

The reaction rate constants are given by

$$k = k_1(T_1) \exp \left[\frac{E}{R} \left(\frac{1}{T_1} - \frac{1}{T} \right) \right], \quad K_C = K_{C2}(T_2) \exp \left[\frac{\Delta H_{R_x}^{\circ}}{R} \left(\frac{1}{T_2} - \frac{1}{T} \right) \right]$$

where $k_1(T_1)$ and $K_{C2}(T_2)$ are values of k and K_C at T_1 and T_2 , respectively. For a gas-phase reaction, $\epsilon = 0$, and the concentration of each species can be expressed as

$$C_A = C_{A0} \left(1 - X \right) \frac{T_0}{T}, \quad C_B = C_{A0} \left(\Theta_B - X \right) \frac{T_0}{T}, \quad C_C = 2C_{A0}X \frac{T_0}{T}$$

or

$$C_A = C_{T0} \frac{F_A}{F_T} \frac{T_0}{T}, \quad C_B = C_{T0} \frac{F_B}{F_T} \frac{T_0}{T}, \quad C_C = C_{T0} \frac{F_C}{F_T} \frac{T_0}{T}$$

where T_0 is the feed temperature and $F_T = F_A + F_B + F_C$.

The energy balance yields

$$\frac{dT}{dV} = \frac{Ua(T_a - T) + (-r_A)(-\Delta H_{R_x})}{F_{A0} [C_{pA} + \Theta_B C_{pB} + X \Delta C_p]} = \frac{Ua(T_a - T) + (-r_A)(-\Delta H_{R_x})}{F_A C_{pA} + F_B C_{pB} + F_C C_{pC}}$$

If the heat transfer fluid (coolant) temperature, T_a , is not constant, the energy balance on the heat exchange fluid gives

$$\text{Cocurrent flow: } \frac{dT_a}{dV} = \frac{Ua(T - T_a)}{\dot{m}_c C_{pc}}$$

$$\text{Countercurrent flow: } \frac{dT_a}{dV} = \frac{Ua(T_a - T)}{\dot{m}_c C_{pc}}$$

where

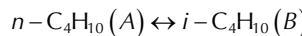
U is the overall heat transfer coefficient

\dot{m}_c is the mass flow rate of the heat transfer fluid

C_{pc} is the heat capacity of the heat transfer fluid

Example 6.23: Isomerization of *n*-butane³⁰

The isomerization reaction of *n*-butane (C_4H_{10}) to isobutane



is to be carried out in a bank of 10 tubular reactors; each reactor is $V = 5 \text{ m}^3$. The bank reactors are double-pipe heat exchangers with the reactants flowing in the inner pipe and $Ua = 5000 \text{ kJ}/(\text{m}^3 \cdot \text{hr} \cdot \text{K})$.

A mixture of 90 mol % *n*-butane and 10 mol % *i*-pentane, which is considered an inert, is to be processed at 70% conversion. The molar flow rate of the mixture is 163 kmol/hr. The bank reactors can be considered as a countercurrent heat exchanger. The entering temperature of the reactants is $T_0 = 305 \text{ K}$ and the entering coolant temperature is $T_a = 310 \text{ K}$. For a countercurrent heat exchanger, this value is the entering coolant temperature $T_{a0} (= 310 \text{ K})$ at $V = V_{final} = 5 \text{ m}^3$. In order to find the coolant temperature at the outlet ($V = 0$), we guess T_a at $V = 0$ and see if it matches T_{a0} at $V = V_{final} = 5 \text{ m}^3$. If it doesn't match, we guess again.

The mass flow rate of the coolant is $\dot{m}_c = 500 \text{ kg/hr}$ and the heat capacity of the coolant is $C_{pc} = 28 \text{ kJ/kg/K}$. The temperature in any one of the reactors cannot rise above 325 K. At $T_1 = 360 \text{ K}$ and $T_2 = 333 \text{ K}$, it is known that $k_1(T_1) = 31.1 \text{ hr}^{-1}$ and $K_{C2}(T_2) = 3.03 \text{ hr}^{-1}$. Plot conversion X , equilibrium conversion X_e , temperature T , and reaction rate $-r_A$ down the length of the reactor.

Data: $\Delta H_{R_x}^o = -6900 \text{ J/mol}$, activation energy $E = 65,700 \text{ J/mol}$, $C_{A0} = 9.3 \text{ kmol/m}^3$,

$$R = 8.314 \text{ J}/(\text{mol} \cdot \text{K}), C_{pA} = C_{pB} = 141 \text{ J}/(\text{mol} \cdot \text{K}), C_{p_{i-p}} = 161 \text{ J}/(\text{mol} \cdot \text{K}).$$

Solution

For each reactor, $F_{A0} = 0.9F_0 = (0.9)(163) \times \frac{1}{10} = 14.67 \text{ kmolA/hr}$, $T_0 = 305 \text{ K}$

$$C_{p0} = \sum \Theta_i C_{pi} = C_{pA} + \Theta_i C_{pI} = \left(141 + \frac{0.1}{0.9} 161 \right) = 158.889 \text{ J}/(\text{mol} \cdot \text{K})$$

$$\Delta C_p = C_{pB} - C_{pA} = 141 - 141 = 0$$

The concentration of each species can be expressed as

$$C_A = C_{A0} (1 - X_0), \quad C_B = C_{A0} X$$

Substitution of these relations to the rate equation gives

$$-r_A = k C_{A0} \left[1 - \left(1 + \frac{1}{K_c} \right) X \right]$$

The rate of change of conversion with respect to reactor volume is given by

$$\frac{dX}{dV} = \frac{-r_A}{F_{A0}}$$

At equilibrium, $-r_A = 0$, and the equilibrium conversion is given by

$$X_e = \frac{K_c}{1 + K_c}$$

The energy balance yields

$$\frac{dT}{dV} = \frac{Ua(T_a - T) + r_A \Delta H_{R_A}}{F_{A0} \sum \Theta_i C_{pi}} = \frac{r_A \Delta H_{R_A} - Ua(T - T_a)}{F_{A0} C_{p0}}$$

$$\text{For a countercurrent heat exchanger, } \frac{dT_a}{dV} = \frac{Ua(T_a - T)}{\dot{m}_c C_{pc}}.$$

We guess $T_a(V=0) = T_{a0}$ and see if it matches $T_a = 310$ K at $V = V_{final} = 5$ m³.

The differential equations are defined by the function `exbco`. The input argument `hx` denotes the type of a heat exchanger: for countercurrent heat exchanger, `hx = 'cn'`, for cocurrent heat exchanger, `hx = 'co'`, and for constant T_a , `hx = 'ct'`.

```
function dz = exbco(V, z, Ca0, Fa0, Cp0, Cpc, Ua, m, T1, T2, k1, K2, E, R, dH, hx)
% z(1)=Ta, z(2)=X, z(3)=T
k = k1*exp(E*(1/T1 - 1/z(3))/R); Kc = K2*exp(dH*(1/T2 - 1/z(3))/R);
ra = -k*Ca0*(1 - (1 + 1/Kc)*z(2));
if hx == 'co'
    dz(1) = Ua*(z(3) - z(1))/(m*Cpc);
elseif hx == 'cn'
    dz(1) = -Ua*(z(3) - z(1))/(m*Cpc);
else
    dz(1) = 0;
end
dz(2) = -ra/Fa0;
dz(3) = (ra*dH - Ua*(z(3)-z(1)))/(Fa0*Cp0);
dz = dz';
end
```

The script `useexbco` sets data, calls the function `exbco`, and employs the built-in solver `ode45` to solve the system of differential equations. For a countercurrent heat exchanger, the initial value of T_a at $V = 0$ is unknown. T_a is updated iteratively until the temperature at $V = V_{final} = 5$ m³ is satisfied by using a numerical analysis method such as the bisection scheme. In this script, the initial search region of $305 \leq T_a \leq 320$ is used.

```
% useexbco.m
clear all;
Ca0 = 9.3; Fa0 = 14.67; T1 = 360; T2 = 333; k1 = 31.1; K2 = 3.03;
```

```

E = 65700; R = 8.314; dH = -6900; Cp0 = 158.889; Cpc = 28;
Ua = 5000; m = 500; hx = 'cn'; crit = 1e-3; Taf = 310; errT = 10;
Vi = [0 5];
if hx == 'cn' % guess Ta(z1)
    Ta01 = 305; Ta02 = 320; Ta0m = (Ta01+Ta02)/2;
    while errT >= crit
        z01 = [Ta01 0 305]; z02 = [Ta02 0 305]; z0m = [Ta0m 0 305];
        [V z1] = ode45(@exbco,Vi,z01,[],Ca0,Fa0,Cp0,Cpc,Ua,m, ...
            T1,T2,k1,K2,E,R,dH,hx);
        [V zm] = ode45(@exbco,Vi,z0m,[],Ca0,Fa0,Cp0,Cpc,Ua,m, ...
            T1,T2,k1,K2,E,R,dH,hx);
        [V z2] = ode45(@exbco,Vi,z02,[],Ca0,Fa0,Cp0,Cpc,Ua,m, ...
            T1,T2,k1,K2,E,R,dH,hx);
        if (z1(end,1)-Taf)*(zm(end,1)-Taf) < 0
            Ta02 = Ta0m; Ta0m = (Ta01+Ta02)/2;
        else
            Ta01 = Ta0m; Ta0m = (Ta01+Ta02)/2;
        end
        errT = abs(zm(end,1) - Taf);
    end
else
    z0 = [310 0 305];
    [V zm] = ode45(@exbco,Vi,z0,[],Ca0,Fa0,Cp0,Cpc,Ua,m, ...
        T1,T2,k1,K2,E,R,dH,hx);
end
Ta = zm(:,1); X = zm(:,2); T = zm(:,3);
k = k1*exp(E*(1/T1 - 1./T)/R); Kc = K2*exp(dH*(1/T2 - 1./T)/R);
ra = -k*Ca0.* (1 - (1 + 1./Kc).*X); Xe = Kc./(1+Kc);
figure(1)
subplot(1, 2), plot(V,T,V,Ta,'--'), xlabel('V'), ylabel('T(K)'), ...
    legend('T','T_a')
subplot(1, 2), plot(V,X,V,Xe,'--'), xlabel('V'), ylabel('X,X_e'), ...
    legend('X','Xe')
figure(2)
plot(V,-ra), xlabel('V'), ylabel('-r_A')

```

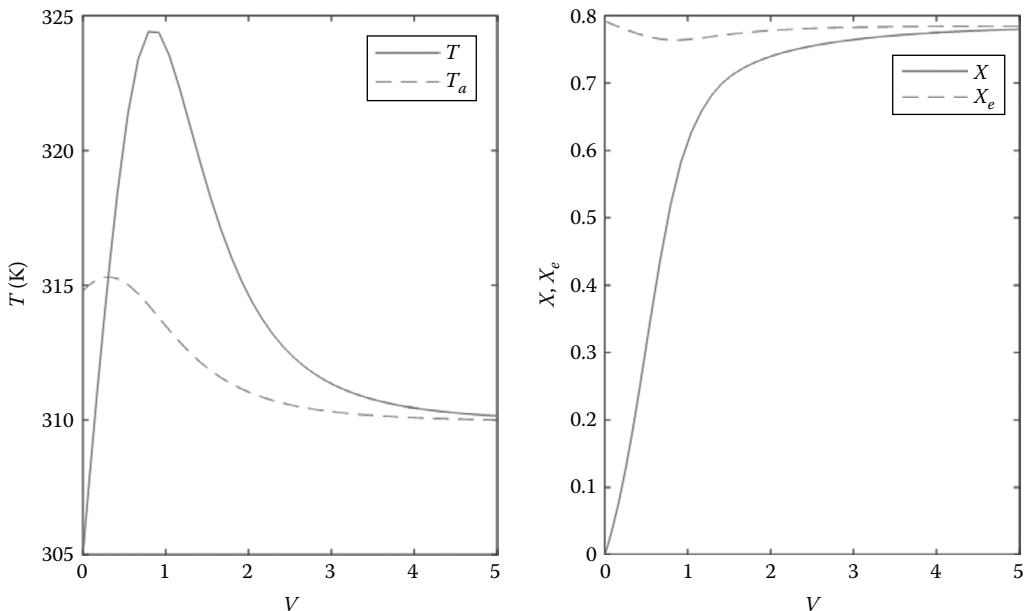
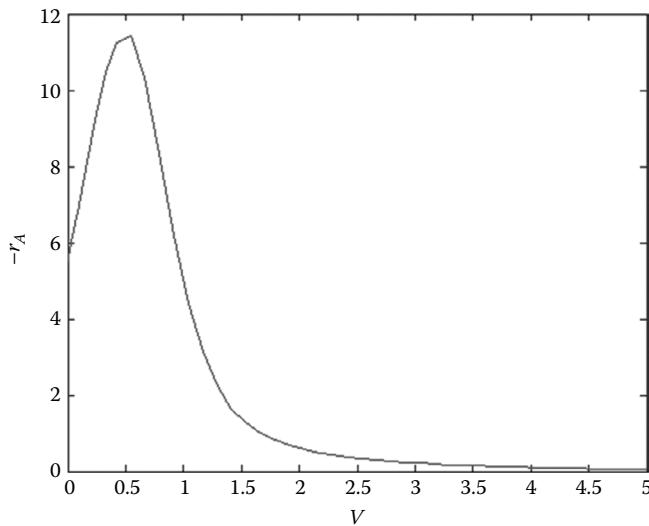


FIGURE 6.27 Temperature and conversion profile.

**FIGURE 6.28** Reaction rate profile.

```

fprintf('Conversion (X) and equilibrium conversion (Xe): Xf = %g,
Xef = %g\n',X(end),Xe(end));
fprintf('Final T and Ta: Tf = %g, Taf = %g\n',T(end), Ta(end));
fprintf('Final reaction rate: raf = %g\n',-ra(end));

```

The script *useexbco* produces the following outputs and [Figures 6.27](#) and [6.28](#):

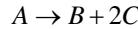
```

>> useexbco
Conversion (X) and equilibrium conversion (Xe): Xf = 0.779843, Xef = 0.784512
Final T and Ta: Tf = 310.153, Taf = 310.001
Final reaction rate: raf = 0.0505495

```

PROBLEMS

6.1 A homogeneous irreversible gas-phase reaction whose stoichiometry is represented by



is carried out in a CSTR with volume $V = 1 \text{ dm}^3$ at 300°C and 0.9125 atm . The data for conversion, X_A , versus the feed flow rate v_0 (dm^3/sec), at reactor conditions where the reactor feed consists of pure reactant A are shown in [Table P6.1](#).³¹ A mass balance on reactant A for this reactor yields

$$v_0 = \frac{kVC_{A0}^{n-1} (1+X_A)^n}{X_A (1+2X_A)^n}$$

TABLE P6.1
Conversion versus Feed Flow Rate

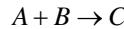
v_0	250	100	50	25	10	5	2.5	1	0.5
X_A	0.45	0.5562	0.6434	0.7073	0.7874	0.8587	0.8838	0.9125	0.95

TABLE P6.2
Concentration Data

t (min)	0	50	100	150	200	250	300
C_A (mol/dm ³)	0.05	0.038	0.0306	0.0256	0.0222	0.0195	0.0174

Estimate the reaction order n with respect to A and the corresponding value of the reaction rate coefficient k . The initial concentration of A is $C_{A0} = 0.1942$ gmol/dm³.

6.2 The liquid-phase reaction



was carried out in a batch reactor. The concentration of the reactant A was measured as a function of time and is shown in [Table P6.2](#).

The concentration of B is assumed to be constant. Use the integral method to confirm that the reaction is 2nd-order with regard to the species A . Calculate the reaction rate constant.³²

6.3 The liquid-phase reaction



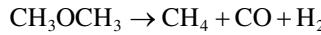
takes place in a batch reactor. The reaction rate can be expressed by

$$-r_A = k' C_A^\alpha$$

The concentration of the reactant A is measured as a function of time and is shown in [Table P6.3](#).

The concentration of B is assumed to be constant. Use the differential method to calculate the reaction order α and the reaction rate constant k' .³³

6.4 The gas-phase 1st-order decomposition reaction of dimethyl ether



is carried out in a constant-volume batch reactor at 552°C. A mass balance on the batch reactor for the 1st-order reaction gives

$$\ln\left(\frac{3P_0 - P}{2P_0}\right) = -kt$$

where

P_0 is the initial pressure

P is the measured pressure

k is the rate constant

[Table P6.4](#) shows pressure changes in the decomposition reaction. Determine the 1st-order rate constant from the data given in [Table P6.4](#).³⁴

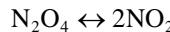
TABLE P6.3
Concentration Data

t (min)	2	48	95	152	196	245	300
C_A (mol/dm ³)	0.035	0.033	0.030	0.025	0.022	0.019	0.017

TABLE P6.4
Pressure Changes in the Decomposition of Dimethyl Ether

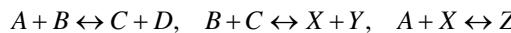
<i>t</i> (sec)	<i>P</i> (mmHg)	<i>t</i> (sec)	<i>P</i> (mmHg)
0	420	182	891
57	584	219	954
85	662	261	1013
114	743	299	1054
145	815		

- 6.5** The reversible gas-phase decomposition of nitrogen tetroxide, N_2O_4 , to nitrogen dioxide, NO_2 ,



is to be carried out at constant temperature. The feed consists of pure N_2O_4 at 340 K and 202.6 kPa (2 atm). The concentration equilibrium constant, K_C , at 340 K is 0.1 mol/dm³ and the rate constant is $K_{\text{N}_2\text{O}_4} = 0.5 \text{ min}^{-1}$. Calculate the equilibrium conversion of N_2O_4 in a constant-volume batch reactor.³⁵ The concentration equilibrium constant is given by $K_C = \frac{C_{\text{Be}}^2}{C_{\text{Ae}}}$.

- 6.6** Gas-phase reactions are taking place in a constant-volume batch reactor:



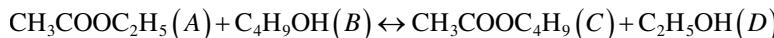
The nonlinear equilibrium relationships are expressed as

$$K_1 = \frac{C_C C_D}{C_A C_B}, \quad K_2 = \frac{C_X C_Y}{C_B C_C}, \quad K_3 = \frac{C_Z}{C_A C_X}$$

$$C_A = C_{A0} - C_D - C_Z, \quad C_B = C_{B0} - C_D - C_Y, \quad C_C = C_D - C_Y, \quad C_Y = C_X + C_Z$$

The initial concentrations of A and B are $C_{A0} = C_{B0} = 1.5$ (mol/liter), and the equilibrium constants are $K_1 = 1.06$, $K_2 = 2.63$, $K_3 = 5$. Calculate the concentration of each species at equilibrium when the initial estimates of D, X, and Z are $C_D = C_X = C_Z = 0$ (mol/liter).³⁶ Can we get feasible solutions for nonzero initial estimates such as $C_D = C_X = C_Z = 1$?

- 6.7** Pure butanol is to be fed into a semibatch reactor containing pure ethyl acetate to produce butyl acetate and ethanol. The reaction can be expressed as



The mole balance, rate law, and stoichiometry equations are as follows:

$$\text{Mole balance: } \frac{dN_A}{dt} = r_A V, \quad \frac{dN_B}{dt} = r_A V + v_0 C_{B0}, \quad \frac{dN_C}{dt} = -r_A V, \quad \frac{dN_D}{dt} = -r_A V$$

$$\text{Rate law: } -r_A = k \left(C_A C_B - \frac{C_C C_D}{K_e} \right)$$

$$\text{Stoichiometry: } C_A = \frac{N_A}{V}, \quad C_B = \frac{N_B}{V}, \quad C_C = \frac{N_C}{V}, \quad C_D = \frac{N_D}{V}$$

The overall mass balance and conversion are given by

$$\frac{dV}{dt} = v_0, \quad x_A = \frac{N_{A0} - N_A}{N_{A0}}$$

respectively. The reaction is carried out at 300 K. At this temperature, the equilibrium constant based on concentrations is $K_e = 1.08$ and the reaction rate constant is $k = 9 \times 10^{-5} \text{ dm}^3/\text{gmol}$. Initially, there are $V(0) = 200 \text{ dm}^3$ of ethyl acetate in the reactor, and butanol is fed at a rate of $v_0 = 0.05 \text{ dm}^3/\text{sec}$ for a period of 4000 sec from the start of reactor operation. The initial concentrations of ethyl acetate in the reactor and butanol in the feed stream are $C_{A0} = 7.72 \text{ gmol/dm}^3$ and $C_{B0} = 10.93 \text{ gmol/dm}^3$, respectively.³⁷ Plot the conversion x_A of ethyl acetate versus time for the first 5000 sec of reactor operation.

6.8 Complex liquid-phase irreversible reactions



are taking place in a semibatch reactor. Initially, the reactor contains the component B , the initial concentration of which is $C_{B0} = 0.2 \text{ mol/dm}^3$. The reactant A is fed to B in the reactor at a rate of $F_{A0} = 3 \text{ mol/min}$. The volumetric flow rate of the reactant is $10 \text{ dm}^3/\text{min}$ and the initial reactor volume is $V = 1000 \text{ dm}^3$. The reaction rate for each species can be expressed as

$$r_A = -k_a C_A C_B^2 - \frac{2}{3} k_c C_A^2 C_C^3, \quad r_B = -2k_a C_A C_B^2, \quad r_C = k_a C_A C_B^2 - k_c C_A^2 C_C^3, \quad r_D = \frac{1}{3} k_c C_A^2 C_C^3$$

where $C_i = \frac{N_i}{V}$ ($i = A, B, C, D$) and $V = V_0 + v_0 t$. The mole balances gives

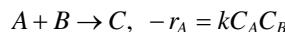
$$\frac{dN_A}{dt} = r_A V + F_{A0}, \quad \frac{dN_B}{dt} = r_B V, \quad \frac{dN_C}{dt} = r_C V, \quad \frac{dN_D}{dt} = r_D V$$

The selectivity is defined by

$$S_{C/D} = \frac{N_C}{N_D}$$

Plot the number of moles of each species and the selectivity as functions of time for the first 100 min ($0 \leq t \leq 100 \text{ (min)}$) of reactor operation.³⁸

6.9 The elementary irreversible liquid-phase reaction



is carried out in a series of three identical CSTRs. The initial concentrations of A and B are $C_{A0} = C_{B0} = 2 \text{ gmol/dm}^3$, the inlet flow rates of A and B are $v_{0A} = v_{0B} = 6 \text{ dm}^3/\text{min}$, the reaction rate constant is $k = 0.5$, the volume of each reactor is $V_1 = V_2 = V_3 = 200 \text{ dm}^3$, and the flow rate of each stream is $v_1 = v_2 = v_3 = 12 \text{ dm}^3/\text{min}$, respectively.

1. Calculate the steady-state concentrations of A and B in each reactor.
2. Plot the concentration of B exiting each reactor, C_{Bi} ($i = 1, 2, 3$), during start-up to the final time $t = 20$ ($0 \leq t \leq 20$).

6.10 Methane and water are formed from carbon monoxide and hydrogen using a nickel catalyst. Table P6.10 shows calculated reaction rates.

TABLE P6.10
Calculated Reaction Rates for Methane Formation Reaction

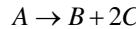
P_{CO} (atm)	P_{H_2} (atm)	r'_{CH_4} (Reaction rate)
1.0	1.0	5.20×10^{-3}
1.8	1.0	13.2×10^{-3}
4.08	1.0	30.0×10^{-3}
1.0	0.1	4.95×10^{-3}
1.0	0.5	7.42×10^{-3}
1.0	4.0	5.25×10^{-3}

The rate equation can be expressed as

$$r'_{CH_4} = \frac{aP_{CO}P_{H_2}^{\beta_1}}{1 + bP_{H_2}^{\beta_2}}$$

Estimate the parameters a , b , β_1 , and β_2 .³⁹

- 6.11** The irreversible 1st-order decomposition of the di-tert-butyl peroxide is to be carried out in an isothermal plug-flow reactor in which there is no pressure drop. The reaction can be expressed as



The reaction rate constant for this 1st-order reaction is $k = 0.08 \text{ min}^{-1}$. The reactor volume is 200 dm^3 , and the entering volumetric flow rate is maintained constant at $10 \text{ dm}^3/\text{min}$. The feed stream consists of pure A at a concentration of $C_{A0} = 1.0 \text{ gmol/dm}^3$. Plot the conversion X as a function of reactor volume.⁴⁰

- 6.12** The irreversible 1st-order decomposition of the di-tert-butyl peroxide is to be carried out in an isothermal plug-flow reactor in which there is no pressure drop. The reactor volume is 200 dm^3 , and the entering volumetric flow rate is maintained constant at $10 \text{ dm}^3/\text{min}$. The initial concentration of reactant A is $C_{A0} = 1.0 \text{ gmol/dm}^3$.⁴⁰
1. The reaction can be expressed as $A \rightarrow B + 2C$ and the reaction rate constant for this 1st-order reaction is $k = 0.08 \text{ min}^{-1}$. The feed consists of 5% A and 95% nitrogen gas as an inert component. Plot the conversion X as a function of reactor volume.
 2. The reaction can be expressed as $3A \rightarrow B$, and the reaction rate constant for this 1st-order reaction is $k = 0.08 \text{ min}^{-1}$. The feed consists of 5% A and 95% nitrogen gas as an inert component. Plot the conversion X as a function of reactor volume.

- 6.13** An irreversible liquid-phase reaction

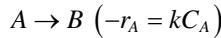


is carried out in a plug-flow reactor. The reaction rate is expressed as

$$-r_A = kC_A^n$$

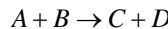
where n is the reaction order and k is the rate constant. The reactor volume is $V = 1.5 \text{ dm}^3$ and the entering volumetric flow rate is maintained constant at $v_0 = 0.9 \text{ dm}^3/\text{min}$. The initial concentration of A is $C_{A0} = 1.0 \text{ gmol/dm}^3$, and the rate constant is $k = 1.1$. Plot the conversion X as a function of reactor volume for $n = 0, 1, 2$, and 3 .

6.14 The catalytic gas-phase 1st-order reaction



is to be carried out in a packed bed reactor under isothermal operation. The reactant is pure A with an inlet concentration of $C_{A0} = 1.0 \text{ gmol/dm}^3$, the entering pressure is 25 atm, and the entering volumetric flow rate is $v_0 = 1 \text{ dm}^3/\text{min}$. The 1st-order reaction rate constant based on reactant A is $k = 1 \text{ dm}^3/(\text{kg cat} \cdot \text{min})$. Plot both the conversion X and relative pressure $y = \frac{P}{P_0}$ as a function of the weight of the packing W ($0 \leq W \leq W_{max} = 2 \text{ kg}$) for the pressure drop parameter $\alpha = 0.12 \text{ kg}^{-1}$.⁴¹

6.15 The irreversible gas-phase catalytic reaction



is to be carried out in a packed bed reactor with four different catalysts (Catalyst 1, Catalyst 2, Catalyst 3, Catalyst 4). For each catalyst, the rate expression has a different form:

$$\text{Catalyst 1: } -r_{A1} = \frac{kC_A C_B}{1 + K_A C_A}$$

$$\text{Catalyst 2: } -r_{A2} = \frac{kC_A C_B}{1 + K_A C_A + K_C C_C}$$

$$\text{Catalyst 3: } -r_{A3} = \frac{kC_A C_B}{(1 + K_A C_A + K_B C_B)^2}$$

$$\text{Catalyst 4: } -r_{A4} = \frac{kC_A C_B}{(1 + K_A C_A + K_B C_B + K_C C_C)^2}$$

The initial concentrations of the reactants are $C_{A0} = C_{B0} = 1.0 \text{ gmol/dm}^3$ at the reactor inlet, and the molar feed flow rate of A is $F_{A0} = 1.5 \text{ gmol/min}$. There is a total of $W_{max} = 2 \text{ kg}$ of each catalyst used in the reactor. The reaction rate constant is $k = 10 \text{ dm}^6/(\text{kg} \cdot \text{min})$, and the various catalytic parameters K_A , K_B , and K_C are given by $K_A = 1 \text{ dm}^3/\text{gmol}$, $K_B = 2 \text{ dm}^3/\text{gmol}$, and $K_C = 20 \text{ dm}^3/\text{gmol}$, respectively. The pressure ratio within the reactor is given by⁴²

$$\frac{dy}{dW} = -\frac{\alpha}{2y}$$

where $y = \frac{P}{P_0}$ and α is a constant ($\alpha = 0.4$). Calculate and plot the conversion X versus catalyst weight W for each of the catalytic rate expressions.

6.16 A gas-phase catalytic reaction



is carried out in a packed bed reactor where the catalyst activity is decaying. The reaction with deactivation follows the rate expression given by

$$-r_A = kaC_A$$

where a is the catalyst activity. The packed bed reactor can be approximated by three CSTRs in series as shown in Figure P6.16.⁴³

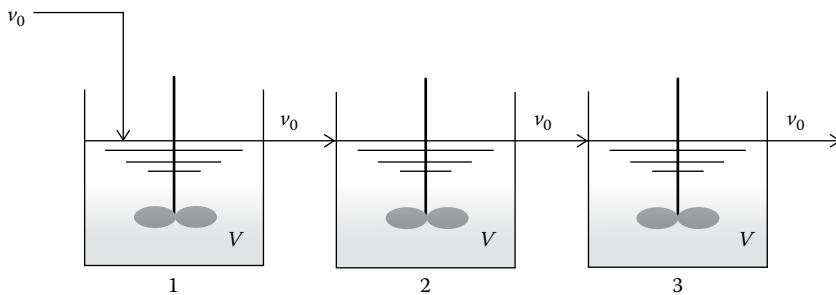


FIGURE P6.16 Train of three CSTRs.

Changes in the concentration of A and B in each of the three reactors can be represented as

$$\frac{dC_{Ai}}{dt} = \frac{v_0}{V} (C_{A(i-1)} - C_{A(i)}) + r_{Ai}, \quad \frac{dC_{Bi}}{dt} = \frac{v_0}{V} (C_{B(i-1)} - C_{B(i)}) - r_{Ai}$$

1. It is assumed that the catalyst activity follows the deactivation kinetics given by

$$\frac{da_i}{dt} = -k_d a_i \quad (i = 1, 2, 3)$$

Plot the concentration of A in each of the three reactors as a function of time to 60 minutes ($0 \leq t \leq 60$).

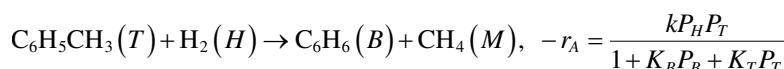
2. It is assumed that the catalyst activity function is given by

$$\frac{da_i}{dt} = -k_d a_i C_{B(i)} \quad (i = 1, 2, 3)$$

Plot the concentration of A in each of the three reactors as a function of time to 60 min ($0 \leq t \leq 60$).

Data: $k_d = 0.01 \text{ min}^{-1}$, $k = 0.9 \text{ dm}^3/(\text{dm}^3(\text{cat}) \cdot \text{min})$, $C_{A0} = 0.01 \text{ gmol/cm}^3$, $C_{B0} = 0$, $v_0 = 5 \text{ dm}^3/\text{min}$, $V = 10 \text{ dm}^3$, $a_i(0) = 1.0 \quad (i = 1, 2, 3)$, $C_{A(i)}(0) = C_{B(i)}(0) = 0 \quad (i = 1, 2, 3)$

- 6.17** The hydrodemethylation of toluene (T) is to be carried out in a packed bed catalytic reactor. In the reactor, toluene is reacted with hydrogen (H) to produce benzene (B) and methane (M).



The molar feed rate of toluene to the reactor is $F_{T0} = 50 \text{ mol/min}$, and the reactor inlet is at 40 atm and 640°C. The feed consists of 30% toluene, 54% hydrogen, and 25% inert. The mole balance gives

$$\frac{dX}{dW} = \frac{-r_T}{F_{T0}}$$

where

X is the conversion

W is the catalyst weight

The partial pressure of each species is given by

$$P_T = P_{T0}(1-X)y, \quad P_H = P_{T0}(\Theta_{H_2} - X)y, \quad P_B = P_{T0}Xy$$

where $\Theta_{H_2} = 0.45/0.30 = 1.5$, $P_{T0} = y_{T0}P_0 = (0.3)(40) = 12 \text{ atm}$, $y = \frac{P}{P_0} = \sqrt{1 - \alpha W}$ and α is the pressure drop parameter. Plot the conversion, the pressure ratio $y = \frac{P}{P_0}$, and the partial pressure of each species as a function of catalyst weight.⁴⁴

- Data: $k = 8.7 \times 10^{-4} \text{ mol}/(\text{atm}^2 \text{ kg}_{\text{cat}} \text{ min})$, $K_B = 1.39 \text{ atm}^{-1}$, $K_T = 1.038 \text{ atm}^{-1}$, $\alpha = 9.8 \times 10^{-5} \text{ kg}^{-1}$*
- 6.18** SO_2 is oxidized to form SO_3 in a nonisothermal plug-flow reactor. The conversion can be represented as $X = \frac{C_{\text{SO}_2}(z)}{C_{\text{SO}_2}(z=0)}$. The mass and energy balances on the reactor yield⁴⁵

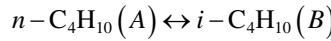
$$\frac{dX}{dz} = -50r, \quad \frac{dT}{dz} = -4.1(T - T_a) + 1.02 \times 10^4 r, \quad X(0) = 1, \quad T(0) = 673.2$$

$$r = \frac{X\sqrt{1 - 0.167(1-X)} - 2.2(1-X)/K_e}{[k_1 + k_2(1-X)]^2}, \quad K_e = -11.02 + \frac{11,570}{T}$$

$$\ln k_1 = -14.96 + \frac{11,070}{T}, \quad \ln k_2 = -1.331 + \frac{2,331}{T}, \quad T_a = 673.2$$

Plot the profiles of X and T as a function of dimensionless reactor length (z).

- 6.19** The isomerization reaction of *n*-butane (C_4H_{10}) to isobutane



is to be carried out in a bank of 10 tubular reactors; each reactor is $V = 5 \text{ m}^3$. The bank reactors are double-pipe heat exchangers with the reactants flowing in the inner pipe and $Ua = 5000 \text{ kJ}/(\text{m}^3 \cdot \text{hr} \cdot \text{K})$.

A mixture of 90 mol % *n*-butane and 10 mol % *i*-pentane, which is considered an inert, is to be processed at 70% conversion. The molar flow rate of the mixture is 163 kmol/hr. The bank reactors can be considered as a countercurrent heat changer. The entering temperature of the reactants is $T_0 = 305 \text{ K}$, and the entering coolant temperature is $T_{a0} = 310 \text{ K}$.

The mass flow rate of the coolant is $\dot{m}_c = 500 \text{ kg/hr}$ and the heat capacity of the coolant is $C_{pc} = 28 \text{ kJ/kg/K}$. The temperature in any one of the reactors cannot rise above 325 K. At $T_1 = 360 \text{ K}$ and $T_2 = 333 \text{ K}$, it is known that $k_1(T_1) = 31.1 \text{ hr}^{-1}$ and $K_{C2}(T_2) = 3.03 \text{ hr}^{-1}$. Plot conversion X , equilibrium conversion X_e , temperature T , coolant temperature T_a , and reaction rate $-r_A$ down the length of the reactor for each of the following cases⁴⁶.

1. Cocurrent heat exchange

2. Constant T_a

Data: $\Delta H_{R_x}^\circ = -6,900 \text{ J/mol}$, activation energy $E = 65,700 \text{ J/mol}$, $C_{A0} = 9.3 \text{ kmol/m}^3$

$$R = 8.314 \text{ J}/(\text{mol} \cdot \text{K}), \quad C_{pA} = C_{pB} = 141 \text{ J}/(\text{mol} \cdot \text{K}), \quad C_{R_i-P} = 161 \text{ J}/(\text{mol} \cdot \text{K})$$

REFERENCES

1. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 90–92, 2011.
2. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 262–263, 2011.
3. Bruce A. F., *Introduction to Chemical Engineering Computing*, John Wiley & Sons, Inc., Hoboken, NJ, pp. 43–47, 2006.
4. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 125–128, 2011.
5. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 462–463, 2008.
6. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education Inc., Boston, MA, pp. 311–312, 2011.
7. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education Inc., Boston, MA, pp. 312–313, 2011.
8. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 465–467, 2008.
9. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, p. 226, 2011.
10. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 227–229, 2011.
11. Fogler, H. Scott, *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 230–232, 2011.
12. Finlayson, B. A., *Introduction to Chemical Engineering Computing*, John Wiley & Sons, Inc., Hoboken, NJ, pp. 118–120, 2006.
13. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 213–216, 2011.
14. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 119–122, 2008.
15. Fogler, H. S., *Elements of Chemical Reaction Engineering*, 3rd ed., Prentice Hall, Boston, MA, p. 523, 1999.
16. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 173, 2008.
17. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 454, 2008.
18. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 451–452, 2011.
19. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 485–486, 2008.
20. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 308–311, 2011.
21. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 183–187, 2011.
22. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 491–492, 2008.
23. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 491–495, 2008.
24. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, p. 218, 2011.
25. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 220–224, 2011.
26. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 317–320, 2011.
27. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, p. 386, 2011.
28. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 387–389, 2011.

29. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 496–500, 2011.
30. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 530–535, 2011.
31. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 476, 2008.
32. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 251–252, 2011.
33. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 255–258, 2011.
34. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 467, 2008.
35. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 125–128, 2011.
36. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 223, 2008.
37. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 458–459, 2008.
38. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 314–315, 2011.
39. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 267–270, 2011.
40. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 445, 2008.
41. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 453, 2008.
42. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 486, 2008.
43. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATHE, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 488–489, 2008.
44. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 453–455, 2011.
45. Finlayson, B. A., *Introduction to Chemical Engineering Computing*, John Wiley & Sons, Inc., Hoboken, NJ, pp. 121–123, 2006.
46. Fogler, H. S., *Essentials of Chemical Reaction Engineering*, Pearson Education International, Boston, MA, pp. 530–535, 2011.

7 Mass Transfer

A major part of the unit operations in the chemical engineering industry involves separating mixtures into their components with changes in stream compositions. These operations are called mass transfer operations, and their driving force is concentration difference (or gradient). Various methods, such as gas adsorption or stripping, distillation, liquid extraction, absorption, drying, or evaporation, may be employed to influence mass transfer operations. These methods utilize differences in vapor pressure or solubility, and not in density or particle sizes. The use of a specific method depends on the properties of the components and whether they exist in solid, liquid, or vapor phase. Again, these separations can be carried out in batch or in continuous systems. Batch operations are always time dependent, while continuous systems often operate under steady-state conditions. A unit operation may be conducted in stage-wise contact, as in plate columns or in a cascade of vessels, or in differential contact as in packed or spray columns. The staged units are designed in terms of the number of theoretical equilibrium stages.

In this chapter, the discussion on calculations of mass transfer operations using MATLAB® is confined to diffusion, evaporation, absorption, distillation, and membrane separation. The main objective of this chapter is to provide readers with knowledge and skills needed for application of MATLAB to handle mass transfer problems. The MATLAB programs presented in this chapter can be used in undergraduate or graduate courses on mass transfer. Researchers and practicing engineers in the field of chemical engineering can apply these MATLAB programs in analysis, design, and simulation of a mass transfer equipment.

7.1 DIFFUSION

7.1.1 ONE-DIMENSIONAL DIFFUSION

Consider binary gas-phase diffusion of component A during evaporation of a pure liquid in a simple diffusion tube. Figure 7.1 shows a cylindrical tube where liquid A is evaporating into a gas mixture of A and B from a liquid layer of pure A near the bottom of the tube.¹

Fick's law for the flux of A can be expressed as

$$\frac{dN_A}{dz} = 0, \quad \frac{dx_A}{dz} = -\frac{(1-x_A)N_A}{D_{AB}C}$$

where

C is the total concentration (kg mol/m³)

D_{AB} is the molecular diffusivity of A in B (m²/sec)

N_A is the mole flux of A (kg mol/(m² · sec))

$x_A = C_A/C$ is the mole fraction of A

At $z = z_1$, x_A is given by $x_A = x_{A1} = P_{A0}/P$. The gas mixture is assumed to be ideal gas. Then, at constant temperature and pressure, the total concentration C ($=P/RT$) and D_{AB} can be considered constant and the concentration profile may be expressed as^{2,3}

$$\frac{1-x_A}{1-x_{A1}} = \left(\frac{1-x_{A2}}{1-x_{A1}} \right)^{(z-z_1)/(z_2-z_1)}, \quad N_{Az} \Big|_{z=z_1} = D_{AB}C \frac{(x_{A1} - x_{A2})}{(z_2 - z_1)(x_B)_{lm}}$$

where

$$(x_B)_{lm} = \frac{x_{B2} - x_{B1}}{\ln(x_{B2}/x_{B1})} = \frac{x_{A1} - x_{A2}}{\ln\left(\frac{1-x_{A2}}{1-x_{A1}}\right)}$$

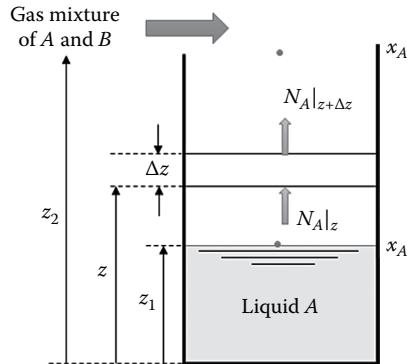


FIGURE 7.1 Gas-phase diffusion of A through gas mixture of A and B. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 384.)

The binary diffusivities for gases are known to vary according to the absolute temperature to the 1.75 power.⁴ If the diffusivity at T_1 is known, the diffusivity at T may be approximated by⁵

$$D_{AB} = D_{AB}|_{T_1} \left(\frac{T}{T_1} \right)^{1.75}$$

7.1.2 MULTICOMPONENT DIFFUSION IN GASES

For multicomponent diffusion in gases at low density to the positive z direction, the Maxwell–Stefan equation⁶ can be expressed as

$$\frac{dC_i}{dz} = \sum_{i=1}^n \frac{x_i N_i - x_j N_j}{D_{ij}}$$

where

C_i (kg mol/m³) is the concentration of species i

x_i is the mole fraction of species i

N_i (kg mol/(m² · sec)) is the mole flux of species i

D_{ij} (m²/sec) is the molecular diffusivity of i in j

n is the number of components

For gas mixture of three components (A, B, C), the Maxwell–Stefan equation yields

$$\begin{aligned} \frac{dC_A}{dz} &= \frac{(x_A N_B - x_B N_A)}{D_{AB}} + \frac{(x_A N_C - x_C N_A)}{D_{AC}} \\ \frac{dC_B}{dz} &= \frac{(x_B N_A - x_A N_B)}{D_{AB}} + \frac{(x_B N_C - x_C N_B)}{D_{BC}} \\ \frac{dC_C}{dz} &= \frac{(x_C N_A - x_A N_C)}{D_{AC}} + \frac{(x_C N_B - x_B N_C)}{D_{BC}} \end{aligned}$$

where $D_{ij} = D_{ji}$ ($i, j = A, B, C$).

Example 7.1: Binary Diffusion⁷

Methanol (A) is evaporated into a stream of dry air (B) in a cylindrical tube at 328.5 K. The distance from the tube inlet to the liquid surface is $z_2 - z_1 = 0.238$ m. At $T = 328.5$ K, the vapor

pressure of methanol is $P_{A0} = 68.4$ kPa and the total pressure is $P = 99.4$ kPa. The binary molecular diffusion coefficient of methanol in air under these conditions is $D_{AB} = 1.991 \times 10^{-5}$ m²/sec.

Calculate the constant molar flux of methanol within the tube at steady state and plot the mole fraction profile of methanol from the liquid surface to the flowing air stream. Compare the calculated molar flux with that obtained from the equation

$$N_{Az} = D_{AB}C \frac{x_0}{(z_2 - z_1)(x_B)_{lm}}, \quad (x_B)_{lm} = \frac{x_0}{\ln(1/(1-x_0))}$$

Solution

At $z = z_1$, $x_{A1} = P/P_{A0}$. We guess N_A , solve the differential equation $dx_A/dz = -(1-x_A)N_A/D_{AB}C$, and check whether the condition $x_{A2} = 0$ is satisfied. If $x_{A2} \neq 0$, the calculation procedure is repeated by using another initial guess of N_A . The bisection method can be effectively used in the iterative calculation procedure. The MATLAB function *xaz* defines the differential equation.

```
function dxdz = xaz(z,x,Na,Dab,C)
dxdz = -(1-x)*Na/Dab/C;
end
```

The script *usexaz* employs the built-in function *ode45* to solve the differential equation and determines N_A by using the bisection method.

```
% usexaz.m
Dab = 1.991e-5; T = 328.5; R = 8.314; P = 99.4; Pa0 = 68.4; Tf = 295;
z2 = 0.238; z1 = 0; zv = [z1 z2];
C = P/(R*T); x0 = Pa0/P; critN = 1e-11; errN = 1;
Na1 = 3.5e-6; Na2 = 3.6e-6;
while errN > critN
    Nam = (Na1+Na2)/2;
    [z x1] = ode45(@xaz,zv,x0,[],Na1,Dab,C);
    [z x2] = ode45(@xaz,zv,x0,[],Na2,Dab,C);
    [z xm] = ode45(@xaz,zv,x0,[],Nam,Dab,C);
    if x1(end)*xm(end) < 0
        Na2 = Nam;
    else
        Na1 = Nam;
    end
    errN = abs(Na1 - Na2);
end
xblm = x0/(log(1/(1-x0)));
Nanal = Dab*C*x0/((z2-z1)*xblm);
fprintf('Estimated Nab = %e, xA = %8.6f\n',Nam, xm(end));
fprintf('Analytic Nab = %e\n',Nanal);
plot(z,xm), xlabel('z(m)'), ylabel('x_A'), axis tight
```

The script *usexaz* produces the following outputs and the plot shown in Figure 7.2:

```
>> usexaz
Estimated Nab = 3.547504e-06, xA = -0.000000
Analytic Nab = 3.547502e-06
```

We can see that $N_{AB} = 3.5475 \times 10^{-6}$ m²/s and $x_{A2} = -4.4158 \times 10^{-7} \approx 0$.

Example 7.2: Multicomponent Diffusion of Gases⁸

Gases A and B are diffusing through stagnant gas C at a temperature of 55°C and a pressure of 0.2 atmospheres from point 1 (z_1) to point 2 (z_2). The distance between these two points is 0.001 m. The molar flux of B was measured to be $N_B = -4.143 \times 10^{-4}$ kg mol/(m²·sec) (i.e., gas B diffuses from z_2 to z_1). The gas mixture is assumed to be ideal gas. Estimate the molar flux of A (N_A).

Data: $C_{A1} = 2.229 \times 10^{-4}$, $C_{A2} = 0$, $C_{B0} = 0$, $C_{B2} = 2.701 \times 10^{-3}$, $C_{C1} = 7.208 \times 10^{-3}$, $C_{C2} = 4.730 \times 10^{-3}$, $C_{AB} = 1.47 \times 10^{-4}$, $C_{AC} = 1.075 \times 10^{-4}$, $D_{BC} = 1.245 \times 10^{-4}$.

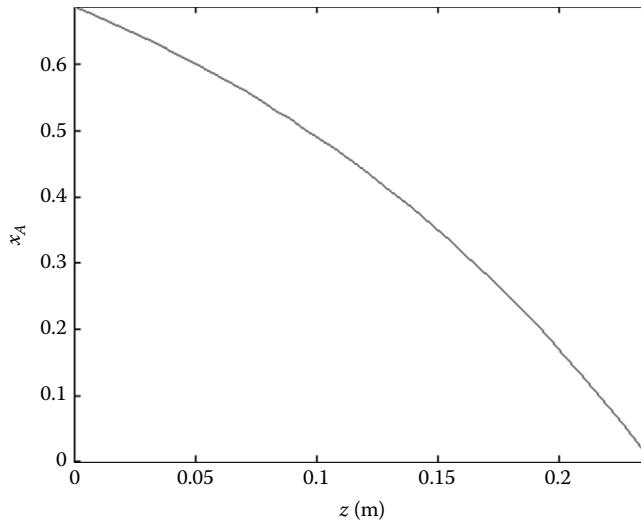


FIGURE 7.2 Mole fraction of methanol along the diffusion path.

Solution

Since component C is stagnant, $N_C = 0$. A simple way to solve this problem is first to assume the value of N_A , solve the differential equations, and check whether the concentration conditions at z_2 are satisfied. The initial guess for N_A can be determined by $N_A = -D_{AC}(C_{A2} - C_{A1})/L$ assuming the gas is a binary mixture of A and C . This procedure is repeated using updated N_A until the concentration conditions at z_2 are satisfied. The bisection method can be used to update N_A . The total concentration is given by $C_t = n/V = P/RT$ from ideal gas law. The MATLAB function *mdif* defines the differential equations.

```
function dcdz = mdif(z,c,D1,D2,D3,Na,Nb,Nc,Ct)
% c1=Ca, c2=Cb, c3=Cc, D1=Dab, D2=Dbc, D3=Dac
xa = c(1)/Ct; xb = c(2)/Ct; xc = c(3)/Ct;
dcdz = [(xa*Nb-xb*Na)/D1 + (xa*Nc-xc*Na)/D3;
(xb*Na-xa*Nb)/D1 + (xb*Nc-xc*Nb)/D2;
(xc*Na-xa*Nc)/D3 + (xc*Nb-xb*Nc)/D2];
end
```

The script *usemdif* employs the built-in function *ode45* to estimate N_A .

```
% usemdif.m
Ca1=2.229e-4; Cb1=0; Cc1=7.208e-3; Ca2=0; Cb2=2.701e-3; Cc2=4.73e-3;
D1=1.075e-4; D2=1.245e-4; D3=1.47e-4;
P=0.2; T=328; R=82.057e-3; Ct = P/(R*T);
L=0.001; Nb=-4.143e-4; Nc=0; Na=-D3*(Ca2-Ca1)/L;
zspan = [0 L]; c0 = [Ca1 Cb1 Cc1]; critN = 1e-10; errA = 1;
Na1 = Na/2; Na2 = 2*Na; iter = 1;
while errA > critN
    Na = (Na1+Na2)/2;
    [z c1] = ode45(@mdif,zspan,[Ca1,Cb1,Cc1],[],D1,D2,D3,Na1,Nb,Nc,Ct);
    [z c2] = ode45(@mdif,zspan,[Ca1,Cb1,Cc1],[],D1,D2,D3,Na2,Nb,Nc,Ct);
    [z cm] = ode45(@mdif,zspan,[Ca1,Cb1,Cc1],[],D1,D2,D3,Na,Nb,Nc,Ct);
    if c1(end,1)*cm(end,1) < 0 % check whether Ca(z2)=0 is satisfied
        Na2 = Na;
    else
        Na1 = Na;
    end
    errA = abs(Na1 - Na2); iter = iter+1;
end
```

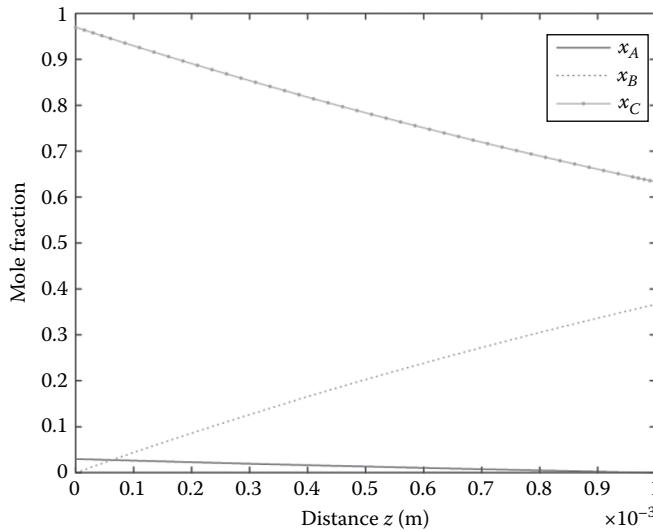


FIGURE 7.3 Mole fraction profiles for components *A*, *B*, and *C*.

```
c = cm; xa = c(:,1)/Ct; xb = c(:,2)/Ct; xc = c(:,3)/Ct;
plot(z,xa,z,xb,'-',z,xc,'.-'), legend('x_A','x_B','x_C')
xlabel('Distance z (m)'), ylabel('Mole fraction')
iter, Na = Nam
```

The script *usemdif* produces the following results and the curves shown in Figure 7.3:

```
>> usemdif
iter =
20
Na =
2.3471e-05
```

We can see that the value of the molar flux of *A* was converged to $N_A = 2.3471 \times 10^{-5} \text{ kg mol}/(\text{m}^2 \cdot \text{sec})$ after 20 iterations.

7.1.3 DIFFUSION FROM A SPHERE

When component *A* diffuses through stagnant fluid *B* from a sphere to a surrounding medium (see Figure 7.4), the mass balance gives

$$\frac{d(N_A r^2)}{dr} = 0, \quad \frac{dp_A}{dr} = -\frac{RTN_A}{D_{AB}} \left(1 - \frac{p_A}{P} \right)$$

where

R is the gas constant ($= 8314.34 \text{ m}^3 \text{Pa}/(\text{kg mol K})$)

T is the absolute temperature (K)

P is the total pressure (Pa)

Integration of this equation yields⁹

$$N_{A1} = \frac{D_{AB}}{RT r_1} \frac{(p_{A1} - p_{A2})}{p_{BM}}, \quad p_{BM} = \frac{p_{A1} - p_{A2}}{\ln \left(\frac{P - p_{A2}}{P - p_{A1}} \right)}$$

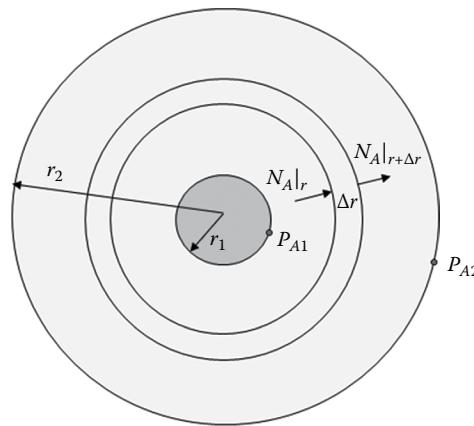


FIGURE 7.4 Diffusion from a sphere. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 392.)

7.1.4 MASS TRANSFER COEFFICIENT

When component *A* is transferred through stagnant medium *B* from the surface of a solid sphere, the mass transfer coefficient of *A* through *B*, k'_c (m/sec), can be represented in terms of the Sherwood number N_{sh} ¹⁰:

$$N_{sh} = k'_c \frac{D_p}{D_{AB}} = 2 + 0.552 N_{re}^{0.53} N_{sc}^{1/3}$$

where

D_p (m) is the diameter of the solid sphere

$N_{re} = D_p v \rho / \mu$ is the Reynolds number

$N_{sc} = \mu / \rho D_{AB}$ is the Schmidt number

ρ (kg/m³) is the gas density

The molar flux of *A* through *B* from a solid sphere, N_A , can be expressed in terms of k'_c :

$$N_A = \frac{k'_c P}{RT} \frac{(p_{A1} - p_{A2})}{p_{BM}}$$

Example 7.3: Diffusion from a Solid Sphere¹¹

Dichlorobenzene (*A*), suspended in stagnant air (*B*), is sublimed at 25°C and atmospheric pressure. The sublimation is taking place at the surface of a sphere of solid dichlorobenzene with a radius of 3×10^{-3} m. The vapor pressure of *A* at 25°C is 1 mmHg, and the diffusivity in air is 7.39×10^{-6} m²/sec. The density of *A* is 1458 kg/m³ and the molecular weight is 147. Calculate the rate of sublimation (flux) and plot the flux as a function of the radius *r*.

Solution

The flux N_A can be determined from

$$\frac{d(N_A r^2)}{dr} = 0, \quad \frac{dp_A}{dr} = -\frac{RT N_A}{D_{AB}} \left(1 - \frac{p_A}{P}\right), \quad N_A = \frac{(N_A r^2)}{r^2}$$

Since $z_2 = p_A$ is the vapor pressure of dichlorobenzene at $r = r_0(p_A(r_0) = 1 \text{ mmHg} = 101,325 \text{ Pa}/760 = 133.32 \text{ Pa})$, $z_2(r_0) = 133.32 \text{ Pa}$. Because the initial condition $z_1(r_0)$ is not given, the initial condition for $z_1 = N_A r^2$ has to be determined such that $p_A = 0$ is satisfied at large r (e.g., $r = 20 \text{ m}$). If x is defined as $r - r_0 = x$, $r = r_0 + x$ and $dr = dx$. The function *spf* defines the differential equations.

```
function dzdx = spf(x,z,T,P,Dab,r0)
R = 8314.34;
dzdx = [0;
-R*T*z(1)*(1-z(2)/P)/(Dab*(x+r0)^2)];
end
```

The script *usespf* uses the built-in function *ode45* to calculate the flux.

```
% usespf.m
Dab = 7.39e-6; T = 298.15; P = 1.01325e5; p0 = 133.32; r0 = 0.003;
xspan = [0 20]; critN = 1e-16; errN = 1;
Na1 = 1e-12; Na2 = 2e-12; % initial guesses
while errN > critN % bisection method
    Nam = (Na1+Na2)/2;
    [x z1] = ode45(@spf,xspan,[Na1,p0],[],T,P,Dab,r0);
    [x z2] = ode45(@spf,xspan,[Na2,p0],[],T,P,Dab,r0);
    [x zm] = ode45(@spf,xspan,[Nam,p0],[],T,P,Dab,r0);
    if z1(end,2)*zm(end,2) < 0
        Na2 = Nam;
    else
        Na1 = Nam;
    end
    errN = abs(Na1 - Na2);
end
r = r0 + x; Na = zm(:,1)./r.^2; pf = zm(end,2);
fprintf('Flux at r=r0: %7.5e, partial pressure at r=inf: %7.5f\n', Na(1), pf);
plot(r(1:25),Na(1:25)), xlabel('r(m)'), ylabel('N_A')
```

The script *usespf* generates the following results and the plot shown in [Figure 7.5](#). We can see that the partial pressure at $r = r_\infty$ is 0.0002 Pa and the molar flux is $N_A|_{r=0} = 1.32575 \times 10^{-7} \text{ kg mol}/(\text{m}^2 \cdot \text{s})$.

```
>> usespf
Flux at r=r0: 1.32575e-07, partial pressure at r=inf: 0.00021
```

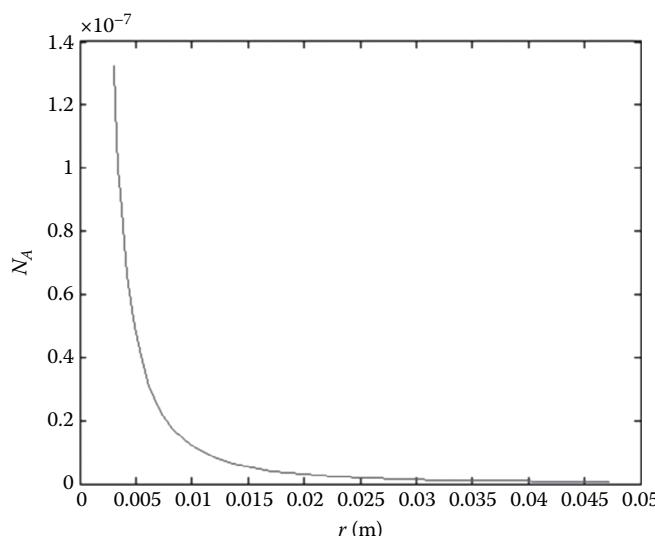


FIGURE 7.5 Molar flux versus radius.

Example 7.4: Drug Delivery by Dissolution of Pill Coating¹²

The pill to deliver a particular drug has a solid spherical inner core of pure drug D and is surrounded by a spherical outer coating of A . The outer coating and the drug dissolve at different rates in the stomach due to their difference in solubility. A person takes all three different pills at the same time. Assume that the stomach is well mixed and that the pills remain in the stomach while they are dissolving.

Let the diameter of pill i be D_i , the diameter of pure drug D in pill i be D_{di} , and the mass transfer coefficient for pill i be k_{Li} ($i = 1, 2, 3$). If the concentration of coating in the stomach is C_{AS} (mg/cm³), the concentration of drug in the stomach is C_{DS} (mg/cm³), the concentration of drug in the body is C_{DB} (mg/kg), and solubilities of outer pill layer and inner drug core at stomach conditions are S_A and S_D (mg/cm³), respectively, the mass balances on volumes of pills yield $dD/dt = -(2k_{Li}/\rho)(S_A - C_{AS})$, $D_1(0) = 0.5$ cm, $D_2(0) = 0.4$ cm, $D_3(0) = 0.35$ cm

$$\frac{dD_i}{dt} = \begin{cases} -\frac{2k_{Li}}{\rho}(S_D - C_{DS}) : & 10^{-5} \leq D_i \leq D_{di} = 0.3 \text{ cm} \\ 0 : & D_i \leq 10^{-5} \text{ cm} \end{cases}$$

$$\frac{dC_{AS}}{dt} = \frac{1}{V} (S_A - C_{AS}) \pi \sum_{i=1}^3 S_{Wi} k_{Li} D_i^2 - \frac{C_{AS}}{\tau}$$

$$\frac{dC_{DS}}{dt} = \frac{1}{V} (S_D - C_{DS}) \pi \sum_{i=1}^3 (1 - S_{Wi}) k_{Li} D_i^2 - \frac{C_{DS}}{\tau}$$

$$k_{Li} = \frac{1.2}{D_i} (i = 1, 2, 3), \quad S_{Wi} = \begin{cases} 1 : D_i > 0.3 \\ 0 : D_i \leq 0.3 \end{cases} \quad (i = 1, 2, 3)$$

where

V (liter) is the volume of fluid in the stomach

τ (hr) is the residence time in the stomach

Plot the diameters of pills (D_1 , D_2 , D_3) and C_{AS} and C_{DS} as a function of time for up to 150 min ($0 \leq t \leq 150$ (min)) after the pills are taken.

Data: $V = 1200$ cm³, $\tau = 240$ min, $S_A = 1$ mg/cm³, $S_D = 0.4$ mg/cm³, $\rho = 1414.7$ mg/cm³

Solution

The system of differential equations is defined by the function *drugf*.

```
function dxdt = drugf(t,x,V,rho,tau,Sa,Sd)
% x(i)=D(i) (i=1,2,3), x(4)=C_AS, x(5)=C_DS
sum0 = 0; sum1 = 0;
for i = 1:3
    kL(i) = 1.2/x(i); Sw(i) = 0;
    if x(i) > 0.3
        fd(i) = -2*kL(i)*(Sa - x(4))/rho; Sw(i) = 1;
    elseif x(i) <= 0.3 && x(i) >= 1e-5
        fd(i) = -2*kL(i)*(Sd - x(5))/rho;
    else
        fd(i) = 0;
    end
    sum0 = sum0 + Sw(i)*kL(i)*x(i);
    sum1 = sum1 + (1-Sw(i))*kL(i)*x(i);
end
```

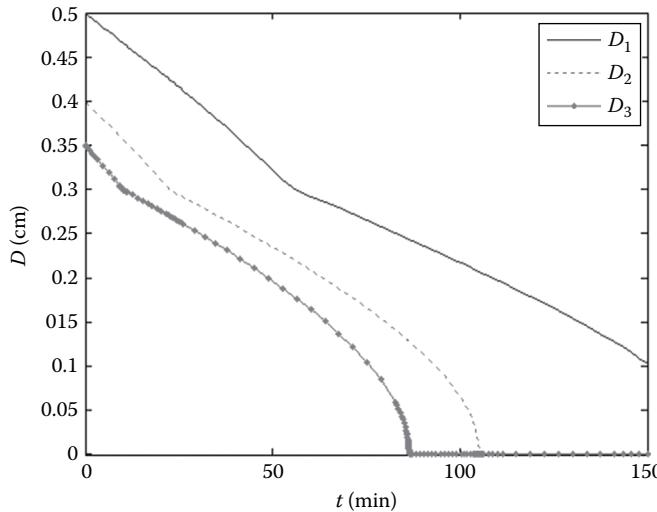


FIGURE 7.6 Diameter variations for three pills.

```

fd(4) = pi*(Sa - x(4))*sum0/V - x(4)/tau;
fd(5) = pi*(Sd - x(5))*sum1/V - x(4)/tau;
dxdt = [fd(1) fd(2) fd(3) fd(4) fd(5)]';
end

```

The script *usedrugf* plots variations of diameters and concentrations as a function of time.

```

% usedrugf.m
x10 = 0.5; x20 = 0.4; x30 = 0.35; x40 = 0; x50 = 0;
V = 1200; tau = 240; Sa = 1; Sd = 0.4; rho = 1414.7;
tspan = [0 150]; x0 = [x10 x20 x30 x40 x50];
[t x] = ode45(@drugf,tspan,x0,[],V,rho,tau,Sa,Sd);
D1 = x(:,1); D2 = x(:,2); D3 = x(:,3); Cas = x(:,4); Cds = x(:,5);
figure(1)
plot(t,D1,t,D2,':',t,D3,'.-'), xlabel('t(min)'), ylabel('D(cm)')
legend('D_1','D_2','D_3'), axis tight
figure(2)
plot(t,Cas,t,Cds,':'), xlabel('t(min)'), ylabel('C(mg/cm^3)')
axis tight, legend('C_{AS}','C_{DS}', 'Location', 'best')

```

The script *usedrugf* generates the plots shown in Figures 7.6 and 7.7.

```
>> usedrugf
```

7.1.5 DIFFUSION IN ISOTHERMAL CATALYST PARTICLES

For a spherical catalyst particle, the mass balance on a differential volume within the particle yields

$$\frac{d}{dr} (N_A r^2) = -k_1 a C_A r^2, \quad N_A = \frac{(N_A r^2)}{r^2}$$

where

N_A is the flux of reactant A

r is the radius of the spherical catalyst particle

k_1 is the 1st-order rate constant based on particle volume

C_A is the concentration of reactant

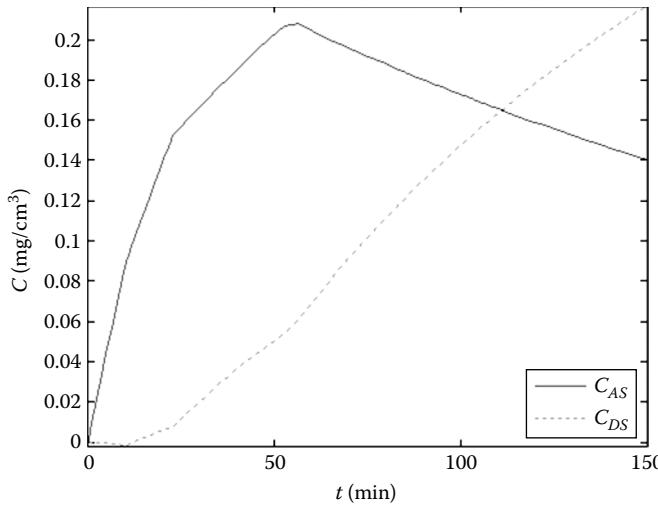


FIGURE 7.7 Concentration variations.

At $r = 0$, $\frac{d}{dr}(N_A r^2) = 0$. Fick's law for the diffusion of reactant A can be expressed as

$$\frac{dC_A}{dr} = -\frac{N_A}{D_e}$$

where D_e is the effective diffusivity for the diffusion of reactant A in the porous particle. The boundary condition for this equation is that the concentration of A at the particle surface is given by $C_A = C_{As}$ when $r = R$. The isothermal internal effectiveness factor, η , is defined as the ratio of the average reaction rate within the particle to the reaction rate at the concentration of the particle surface:

$$\eta = \frac{(-r_A)_{avg}}{(-r_A)_{surface}}$$

η can be calculated from

$$\eta = \frac{\int_0^R k_1 a C_A (4\pi r^2) dr}{k_1 a C_{As} (4\pi R^3 / 3)} = \frac{3}{C_{As} R^3} \int_0^R C_A r^2 dr$$

Differentiation of this equation with respect to r gives

$$\frac{d\eta}{dr} = \frac{3C_A r^2}{C_{As} R^3}, \quad \eta|_{r=0} = 0$$

The analytical solution to the diffusion is given by¹³

$$\eta = \frac{3}{\phi^2} \left\{ \phi \coth(\phi) - 1 \right\}, \quad \phi = R \sqrt{\frac{k_1 a C_{As}^{n-1}}{D_e}}$$

where

ϕ is the Thiele modulus

n is the reaction order

For cylindrical catalyst particles,

$$\frac{d}{dr}(N_A r) = -k_1 a C_A, \quad N_A = \frac{(N_A r)}{r}, \quad \frac{dC_A}{dr} = -\frac{N_A}{D_e}$$

$$\eta = \frac{\int_0^R k_1 a C_A (2\pi r) dr}{k_1 a C_{As} (\pi R^2)} = \frac{2}{C_{As} R^2} \int_0^R C_A r dr$$

$$\frac{d\eta}{dr} = \frac{2C_A r}{C_{As} R^2}, \quad \eta|_{r=0} = 0$$

For the catalytic gas-phase reaction being carried out in a reactor packed with a porous catalyst layer, the mass balance on component A in the z direction from the top of the porous layer yields

$$\frac{dN_A}{dz} = r_A, \quad N_A|_{z=L} = 0$$

where N_A is the molar flux of A . At $z = L$, $N_A = 0$. The rate of change of the concentration of A , C_A , with respect to z is expressed as

$$\frac{dC_A}{dz} = \frac{1}{D_e} [x_A (N_A + N_B) - N_A]$$

where the effective diffusivity of the catalyst layer, D_e , is given by

$$D_e = \frac{D_{AB} \epsilon_p \sigma}{\tilde{\tau}}$$

where

D_{AB} is the binary diffusivity

ϵ_p is the catalyst porosity

σ is the constriction factor

$\tilde{\tau}$ is the tortuosity

Example 7.5: Diffusion with Reaction in Catalyst Particles¹⁴

Calculate the concentration profile for C_A and determine the effectiveness factor η for a 1st-order irreversible reaction in a spherical particle, where $R = 0.5$ cm, $D_e = 0.1$ cm²/sec, $C_{As} = 0.2$ g mol/cm³, and $k_1 a = 6.4$ sec⁻¹.

Solution

Since the initial condition $C_A|_{r=0}$ is unknown, we first assume $C_A|_{r=0}$ and solve the system of differential equations. The initial condition $C_A|_{r=0}$ is updated iteratively until the condition $C_A|_{r=R} = C_{As}$ is satisfied. Once the initial condition is fixed, other values can be calculated. The function $rxnf$ defines the system of differential equations.

```
function dxdr = rxnf(r,x,Cas,R,De,k1a)
% x1=Nar^2, x2=eta, x3=Ca
if r == 0, Na = 0; else Na = x(1)/r^2; end
```

```

dxdr = [-k1a*x(3)*r^2;
3*x(3)*r^2/(Cas*R^3);
-Na/De];
end

```

The script *userxnf* employs the built-in function *ode45* to integrate the system of differential equations and uses the bisection method to determine the initial condition $C_A|_{r=0}$.

```

% userxnf.m
Cas = 0.2; R = 0.5; De = 0.1; k1a = 6.4;
rspan = [0 R]; critN = 1e-6; errN = 1;
Ca1 = 1e-2; Ca2 = 3e-2;
while errN > critN
    Cam = (Ca1+Ca2)/2;
    [r x1] = ode45(@rxnf,rspan,[0,0,Ca1],[],Cas,R,De,k1a);
    [r x2] = ode45(@rxnf,rspan,[0,0,Ca2],[],Cas,R,De,k1a);
    [r xm] = ode45(@rxnf,rspan,[0,0,Cam],[],Cas,R,De,k1a);
    if (x1(end,3)-Cas)*(xm(end,3)-Cas) < 0
        Ca2 = Cam;
    else
        Ca1 = Cam;
    end
    errN = abs(Ca1 - Ca2);
end
Na = xm(:,1)./r.^2; effc = xm(:,2); Ca = xm(:,3);
ephi = R*sqrt(k1a/De); effa = 3*(ephi*coth(ephi)-1)/ephi^2;
fprintf('Effectiveness factor at r=R (calculated): %7.5f\n', effc);
fprintf('Effectiveness factor at r=R (analytic): %7.5f\n', effa);
plot(r,Ca), xlabel('r(cm)'), ylabel('C_A(gmol/cm^3)')

```

The script *userxnf* produces the following outputs and the plot shown in Figure 7.8:

```

>> userxnf
Effectiveness factor at r=R (calculated): 0.56301
Effectiveness factor at r=R (analytic): 0.56300

```

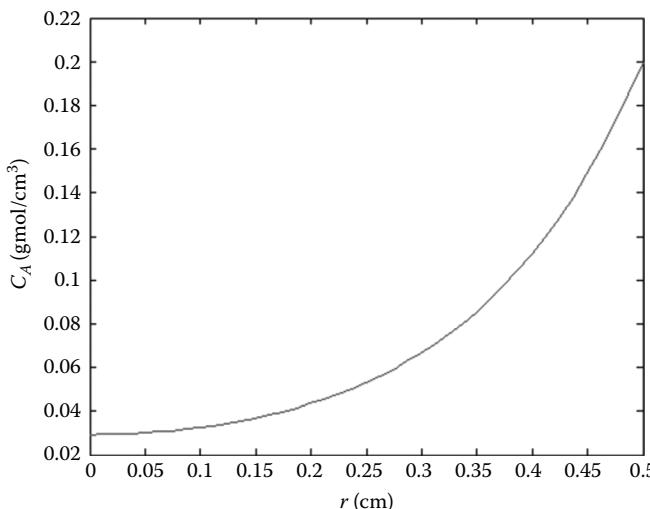


FIGURE 7.8 Concentration profile in a spherical catalyst.

Example 7.6: Reaction and Diffusion in a Porous Catalytic Layer¹⁵

The catalytic gas-phase reversible reaction between components *A* and *B*



is taking place in a porous catalyst layer in a reactor. The reaction rate for reactant *A* is given by

$$r_A = -k \left(C_A^2 - \frac{C_B}{K_c} \right) \text{ g mol / (cm}^3 \cdot \text{sec})$$

where the rate constant $k = 8 \times 10^4 \text{ cm}^3 / (\text{sec} \cdot \text{g mol})$ and the equilibrium constant $K_c = 6 \times 10^5 \text{ cm}^3 / \text{g mol}$. The thickness of the catalytic layer is $L = 0.2 \text{ cm}$, the effective diffusivity of *A* in *B* for this layer is $D_e = 0.01 \text{ cm}^2 / \text{sec}$, the total concentration of *A* and *B* is $C_t = 4 \times 10^{-5} \text{ g mol/cm}^3$, and the concentration of *A* and *B* at the surface of the catalytic layer is $C_{As} = 3 \times 10^{-5} \text{ g mol/cm}^3$ and $C_{Bs} = 1 \times 10^{-5} \text{ g mol/cm}^3$, respectively. Calculate the effectiveness factor for the given reaction (η) and plot C_A and N_A as a function of the depth of the catalytic layer (z).

Solution

Substitution of the relation $C_B = C_t - C_A$ to the rate equation yields

$$\frac{dN_A}{dz} = -k \left(C_A^2 - \frac{C_B}{K_c} \right) = -k \left(C_A^2 - \frac{C_t - C_A}{K_c} \right), \quad N_A|_{z=L} = 0$$

From the reaction stoichiometry, the relation between the molar fluxes of *A* and *B* is given by

$$N_B = -\frac{1}{2} N_A$$

From Fick's law, we have¹⁶

$$\frac{dC_A}{dz} = \frac{1}{D_e} \left[x_A (N_A + N_B) - N_A \right]$$

Substitution of the definition of x_A in terms of concentrations, $x_A = C_A / C_t$, to this equation yields

$$\frac{dC_A}{dz} = \frac{N_A}{2D_e} \left(\frac{C_A}{C_t} - 2 \right), \quad C_A|_{z=0} = C_{As}$$

The effectiveness factor can be expressed as

$$\eta = \frac{(-r_A)|_{\text{avg}}}{(-r_A)|_{\text{surface}}} = \frac{N_{As}/L}{k \left(C_{As}^2 - \frac{C_t - C_{As}}{K_c} \right)}$$

Since the initial condition $N_{As} = N_A|_{z=0}$ is unknown, it should be assumed to solve the differential equations. The initial condition $N_A|_{z=0}$ that satisfies the boundary condition of zero flux at $z = L$, $N_A|_{z=L} = 0$, can be determined by iterative calculations. Once the initial condition is determined, other variables can be calculated. The function *crxf* defines the differential equations.

```
function dxdz = crxf(z,x,Ct,k,Kc,De)
% x1=Na, x2=Ca
dxdz = [-k*(x(2)^2 - (Ct-x(2))/Kc);
x(1)*(x(2)/Ct - 2)/(2*De)];
end
```

The script *usecrxf* determines the initial condition, calculates the effectiveness factor, and generates plots of C_A and N_A versus the depth of the catalytic layer.

```
% usecrxf.m
Cas = 3e-5; Ct=4e-5; L=0.2; De=0.01; k=8e4; Kc=6e5;
zspan = [0 L]; critN = 1e-10; errN = 1;
Na1 = 2e-6; Na2 = 5e-6;
while errN > critN
    Nam = (Na1+Na2)/2;
    [z x1] = ode45(@crxf,zspan,[Na1,Cas],[],Ct,k,Kc,De);
    [z x2] = ode45(@crxf,zspan,[Na2,Cas],[],Ct,k,Kc,De);
    [z xm] = ode45(@crxf,zspan,[Nam,Cas],[],Ct,k,Kc,De);
    if x1(end,1)*xm(end,1) < 0
        Na2 = Nam;
    else
        Na1 = Nam;
    end
    errN = abs(Na1 - Na2);
end
Na = xm(:,1); Ca = xm(:,2);
ras = Cas^2 - (Ct-Cas)/Kc; effc = Na(1)/(L*k*ras);
fprintf('Effectiveness factor: %7.5f\n', effc);
fprintf('Molar flux of A at z=0: %12.8f\n', Na(1));
subplot(1, 2), plot(z,Ca), xlabel('z(cm)'), ylabel('C_A(g mol/cm^3)'), grid
subplot(1, 2), plot(z,Na), xlabel('z(cm)'), ylabel('N_A(g mol/sec/cm^2)'), grid
```

The script *usecrxf* produces the following outputs and the curves shown in [Figure 7.9](#):

```
>> usecrxf
Effectiveness factor: 0.30068
Molar flux of A at z=0: 0.00000425
```

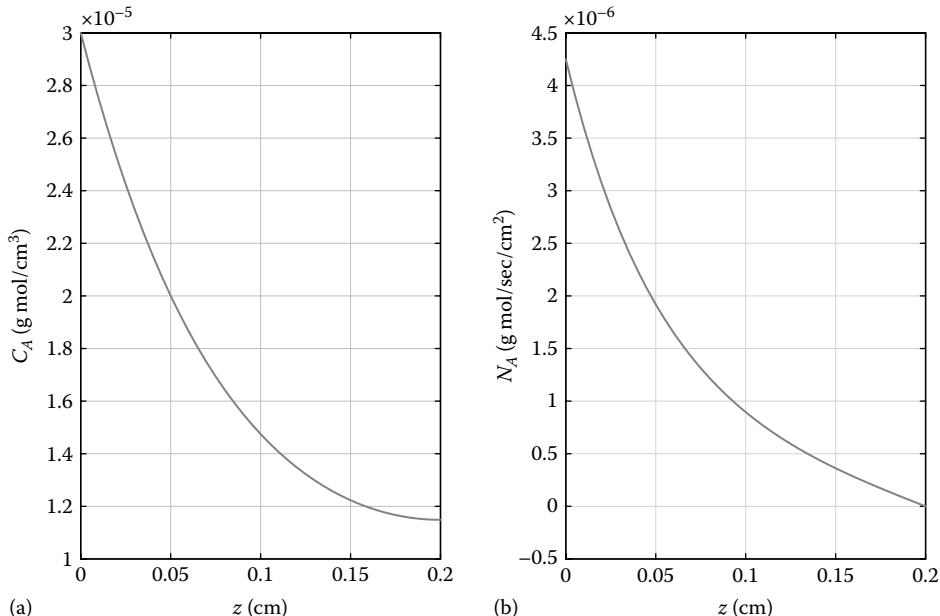


FIGURE 7.9 Concentration and molar flux profiles.

7.2 UNSTEADY-STATE MASS TRANSFER

7.2.1 UNSTEADY-STATE DIFFUSION IN A ONE-DIMENSIONAL SLAB

Unsteady-state diffusion of component A in a one-dimensional slab can be expressed as¹⁷

$$\frac{\partial C_A}{\partial t} = D_{AB} \frac{\partial^2 C_A}{\partial x^2}, \quad C_{A1} \Big|_{x=0} = \frac{C_{A0}}{K}, \quad \frac{\partial C_A}{\partial x} \Big|_{x=L} = 0$$

where

K is the distribution coefficient

D_{AB} is the binary diffusivity

Figure 7.10 shows a one-dimensional slab of width Δx . A central difference formula for the second derivative can be used to approximate the partial derivatives in this equation:

$$\frac{dC_{Ai}}{dt} = \frac{D_{AB}}{(\Delta x)^2} (C_{Ai+1} - 2C_{Ai} + C_{Ai-1}) \quad (2 \leq i \leq n-1)$$

where Δx is the length of the subinterval. At $x = 0$,

$$k_c (C_{A0} - C_{A1}) = -D_{AB} \frac{\partial C_A}{\partial x} \Big|_{x=0} \quad \text{and} \quad \frac{\partial C_A}{\partial x} \Big|_{x=0} = \frac{-C_{A3} + 4C_{A2} - 3C_{A1}}{2\Delta x}$$

and we have

$$C_{A1} = \frac{2k_c C_{A0} \Delta x - D_{AB} C_{A3} + 4D_{AB} C_{A2}}{3D_{AB} + 2k_c K \Delta x}$$

where k_c is the external mass transfer coefficient (m/sec). At $x = L$,

$$\frac{\partial C_A}{\partial x} \Big|_{x=L} = 0 \quad \text{and} \quad \frac{\partial C_{An}}{\partial x} = \frac{3C_{An} - 4C_{An-1} + C_{An-2}}{2\Delta x} = 0$$

This equation can be solved for C_{An} to yield¹⁸

$$C_{An} = \frac{4C_{An-1} - C_{An-2}}{3}$$

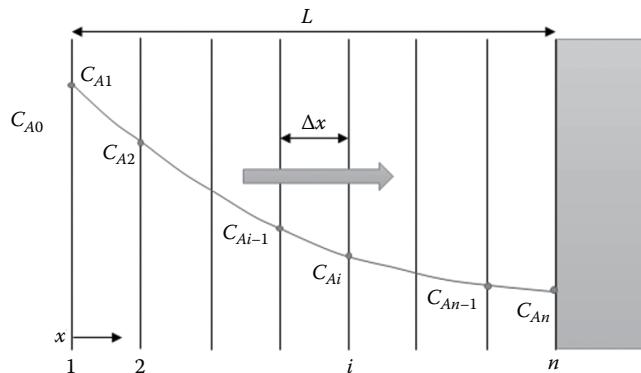


FIGURE 7.10 Unsteady-state diffusion in a one-dimensional slab.

Example 7.7: Unsteady-State Diffusion in a One-Dimensional Slab¹⁹

A slab with a thickness of 0.004 m has one surface suddenly exposed to a solution containing component A with $C_{A0} = 6 \times 10^{-3}$ kg mol/m³. The other surface of the slab is supported by an insulated solid allowing no mass transport. The binary diffusivity is $D_{AB} = 1 \times 10^{-9}$ m²/sec, the distribution coefficient is $K = 1.5$, and the external mass transfer coefficient is $k_c = 1 \times 10^{-6}$ m/sec. At $t = 0$, the concentration of component A at the surface of the solution side is $C_{A1} = 0.001$ kg mol/m³ and that at the solid side is $C_{An} = 0.002$ kg mol/m³. Calculate and plot the concentrations within the slab after 20,000 sec. The interior of the slab may be divided into several intervals with width of each interval being 0.0005 m (i.e., $\Delta x = 0.0005$ m).

Solution

If $\Delta x = 0.0005$ m, $0.0004/0.0005 = 8$ and the number of intervals is $n = 9$. The system of differential equations can be expressed as

$$\frac{dC_{Ai}}{dt} = \frac{D_{AB}}{(\Delta x)^2} (C_{Ai+1} - 2C_{Ai} + C_{Ai-1}) \quad (2 \leq i \leq 8)$$

When $t = 0$, $C_{A1} = 0.001$ kg mol/m³ and $C_{A9} = 0.002$ kg mol/m³, and when $t > 0$, $C_{A1} = C_{A0}/K$ and $C_{A9} = (4C_{A8} - C_{A7})/3$. The initial concentration profile within the slab is assumed to be linear. In this case, the initial concentration at the node i is given by

$$C_{Ai} = C_{A1} + \frac{(C_{A9} - C_{A1})}{8}(i-1) \quad (i = 1, 2, \dots, 9)$$

The function *sldif* defines the system of differential equations.

```
function dxdt = sldif(t,x,Dab,delx,ca0,ca10,ca90,K)
% x(1)=Ca2, x(2)=Ca3, ..., x(7)=Ca8
if t == 0
    ca1 = ca10; ca9 = ca90;
else
    ca1 = ca0/K; ca9 = (4*x(7) - x(6))/3;
end
dxdt = [Dab*(x(2)-2*x(1)+ca1)/(delx^2);
Dab*(x(3)-2*x(2)+x(1))/(delx^2);
Dab*(x(4)-2*x(3)+x(2))/(delx^2);
Dab*(x(5)-2*x(4)+x(3))/(delx^2);
Dab*(x(6)-2*x(5)+x(4))/(delx^2);
Dab*(x(7)-2*x(6)+x(5))/(delx^2);
Dab*(ca9-2*x(7)+x(6))/(delx^2)];
end
```

The script *usesldif* calculates the concentration profile.

```
% usesldif.m
% x(1)=Ca2, x(2)=Ca3, ..., x(7)=Ca8
Dab = 1e-9; delx = 5e-4; ca10 = 1e-3; ca90 = 2e-3; ca0 = 6e-3; K = 1.5;
for k = 1:9
    x0(k) = ca10 + (ca90 - ca10)*(k-1)/8;
end
c0 = [x0(2) x0(3) x0(4) x0(5) x0(6) x0(7) x0(8)];
tf = 20000; tspan = [0 tf];
[t x] = ode45(@sldif,tspan,c0,[],Dab,delx,ca0,ca10,ca90,K);
nt = length(t);
xc1 = [ca10 ones(1,nt-1)*ca0/K]; % Ca1
xc9 = [ca90 ((4*x(2:end,7) - x(2:end,6))/3)']; % Ca9
xc = [];
```

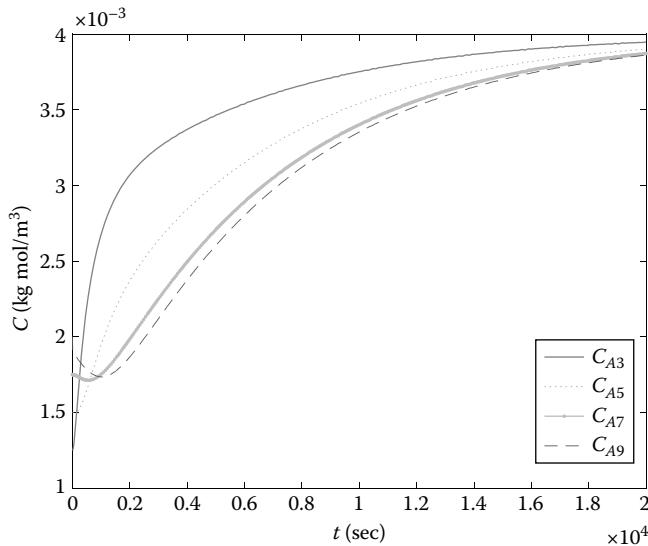


FIGURE 7.11 Concentration profile in a one-dimensional slab.

```

for k = 1:7
    xc = [xc x(:,k)];
end
c = [xc1' xc xc9'];
plot(t,c(:,3),t,c(:,5),':',t,c(:,7),'.-',t,c(:,9), '--')
xlabel('t(sec)'), ylabel('C(kg mol/m^3)')
legend('C_{A3}', 'C_{A5}', 'C_{A7}', 'C_{A9}', 'location', 'best')

```

The script *usesldif* generates the curves shown in Figure 7.11:

```
>> usesldif
```

7.2.2 DIFFUSION IN A FALLING LAMINAR FILM

CO₂ gas is absorbed into a falling liquid film of alkaline solution in which there is a 1st-order irreversible reaction as shown in Figure 7.12. The resulting concentration of the dissolved CO₂ in the film is quite small so that the viscosity of the liquid is not affected. The velocity distribution for the steady-state laminar flow down a vertical wall, v_z (m/sec), is given by²⁰

$$v_z = \frac{\rho g \delta^2}{2\mu} \left[1 - \left(\frac{x}{\delta} \right)^2 \right] = v_{z_{\max}} \left[1 - \left(\frac{x}{\delta} \right)^2 \right]$$

A steady-state mass balance on a differential volume within the liquid film yields the partial differential equation given by²¹

$$v_z \frac{\partial C_A}{\partial z} = D_{AB} \frac{\partial^2 C_A}{\partial x^2} - k' C_A$$

where

C_A (kg mol/m³) is the concentration of dissolved CO₂

D_{AB} (m²/sec) is the diffusivity of dissolved CO₂ in the alkaline solution

k' (sec⁻¹) is a 1st-order reaction rate constant for the neutralization reaction

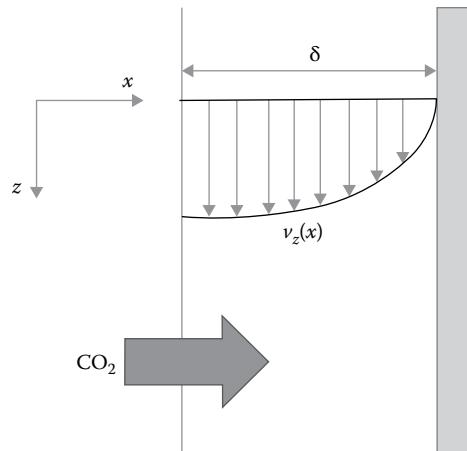


FIGURE 7.12 Mass transfer within a falling laminar film. (From Bird, R.B. et al., *Transport Phenomena*, John Wiley & Sons, Hoboken, NJ, 2002, p. 562.)

The boundary conditions are given by

$$C_A|_{z=0} = 0, \quad C_A|_{x=0, z>0} = C_{As}, \quad \left. \frac{\partial C_A}{\partial x} \right|_{x=\delta, z \geq 0} = 0$$

The partial differential equation can be solved numerically by using the method of lines. The finite difference expressions for the partial differential equation and for the velocity equation can be combined to give

$$\frac{dC_{Ai}}{dz} = \frac{1}{v_{z_{\max}} \left[1 - \left(\frac{(i-1)\Delta x}{\delta} \right)^2 \right]} \left[\frac{D_{AB}}{(\Delta x)^2} (C_{Ai+1} - 2C_{Ai} + C_{Ai-1}) - k' C_{Ai} \right] \quad (2 \leq i \leq n-1)$$

$$C_{Ai}|_{z=0} = 0$$

$$C_{A1} = C_{As}, \quad C_{An} = \frac{4C_{An-1} - C_{An-2}}{3}$$

The finite difference elements are shown in [Figure 7.13](#).

Example 7.8: Mass Transfer in a Falling Laminar Film²²

CO₂ gas is absorbed into a falling liquid film of alkaline solution in which there is no reaction. The film thickness is $\delta = 3 \times 10^{-4}$ m, the maximum velocity is $v_{z_{\max}} = 0.6$ m/sec, and the diffusivity of dissolved CO₂ in the alkaline solution is $D_{AB} = 1.5 \times 10^{-9}$ m²/sec. Use the numerical method of lines with 10 intervals to calculate the concentration of dissolved CO₂ at $z = 1$ m (see [Figure 7.13](#)).

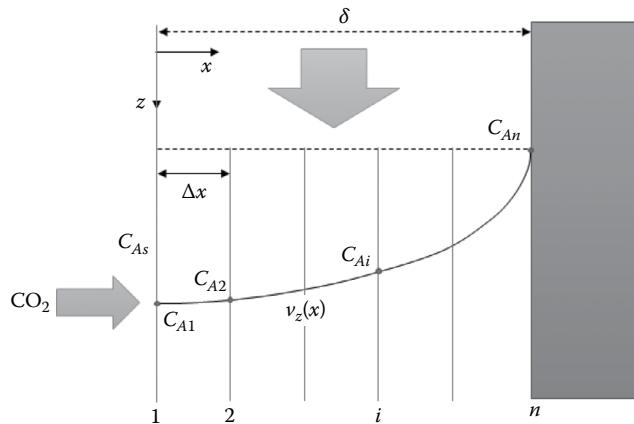


FIGURE 7.13 Mass transfer within a falling laminar film divided into small intervals. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 440.)

Solution

Since the liquid film is divided into 10 intervals, $n = 11$ and we have 9 differential equations.

$$\frac{dC_{Ai}}{dz} = \frac{1}{V_{z_{\max}} \left[1 - \left(\frac{(i-1)\Delta x}{\delta} \right)^2 \right]} \left[\frac{D_{AB}}{(\Delta x)^2} (C_{Ai+1} - 2C_{Ai} + C_{Ai-1}) - k' C_{Ai} \right] \quad (2 \leq i \leq 10)$$

$C_{Ai}|_{z=0} = 0$, $C_{A1} = C_{As}$, $C_{A11} = (4C_{A10} - C_{A9})/3$. If $4C_{A10} < C_{A9}$, $C_{A11} = 0$.

The function *absf* defines the system of differential equations.

```
function dxdz = absf(z,x,Dab,kp,delx,delt,vm,cas)
% x(1)=Ca2, x(2)=Ca3, ..., x(9)=Ca10
cal = cas;
if (4*x(9) < x(8))
    call1 = 0;
else
    call1 = (4*x(9) - x(8))/3;
end
dxdz = [(Dab*(x(2)-2*x(1)+cal)/(delx^2) - kp*x(1))/(vm*(1 - (1*delx/delt)^2));
(Dab*(x(3)-2*x(2)+x(1))/(delx^2) - kp*x(2))/(vm*(1 - (2*delx/delt)^2));
(Dab*(x(4)-2*x(3)+x(2))/(delx^2) - kp*x(3))/(vm*(1 - (3*delx/delt)^2));
(Dab*(x(5)-2*x(4)+x(3))/(delx^2) - kp*x(4))/(vm*(1 - (4*delx/delt)^2));
(Dab*(x(6)-2*x(5)+x(4))/(delx^2) - kp*x(5))/(vm*(1 - (5*delx/delt)^2));
(Dab*(x(7)-2*x(6)+x(5))/(delx^2) - kp*x(6))/(vm*(1 - (6*delx/delt)^2));
(Dab*(x(8)-2*x(7)+x(6))/(delx^2) - kp*x(7))/(vm*(1 - (7*delx/delt)^2));
(Dab*(x(9)-2*x(8)+x(7))/(delx^2) - kp*x(8))/(vm*(1 - (8*delx/delt)^2));
(Dab*(call1-2*x(9)+x(8))/(delx^2) - kp*x(9))/(vm*(1 - (9*delx/delt)^2))];
```

The script *useabsf* calculates the concentration profile.

```
% useabsf.m
% x(1)=Ca2, x(2)=Ca3, ..., x(9)=Ca10
Dab = 1.5e-9; delt = 3e-4; delx = delt/10; vm = 0.6; cas = 0.03; kp = 0;
c0 = zeros(1, 9); % initial concentration profile
zf = 1; zspan = [0 zf];
```

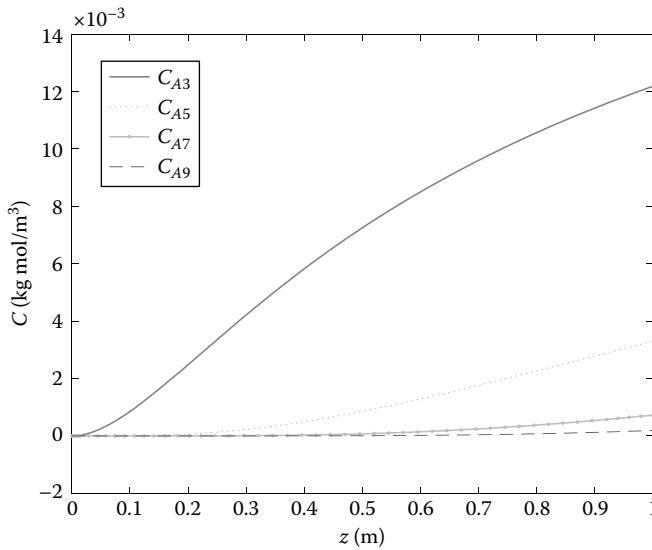


FIGURE 7.14 Concentration profile as a function of distance z .

```

[z x] = ode45(@abesf,zspan,c0,[] ,Dab,kp,delx,delt,vm,cas);
nz = length(z);
xcl = cas*ones(1,nz); % Cal
for k = 1:nz
    if (4*x(k,9) < x(k,8))
        xc11(k) = 0;
    else
        xc11(k) = (4*x(k,9) - x(k,8))/3;
    end
end
xc = [];
for k = 1:9
    xc = [xc x(:,k)];
end
c = [xcl' xc xc11'];
plot(z,c(:,3),z,c(:,5),':',z,c(:,7),'.-',z,c(:,9), '--')
xlabel('z(m)'), ylabel('C(kg mol/m^3)')
legend('C_{A3}', 'C_{A5}', 'C_{A7}', 'C_{A9}', 'location', 'best')

```

The script *useabesf* produces the curves shown in Figure 7.14:

```
>> useabesf
```

7.3 EVAPORATION

7.3.1 SINGLE-EFFECT EVAPORATOR

In evaporation, the vapor from a boiling liquid solution is removed and a more concentrated solution remains. The properties of the solution being concentrated and of vapor being removed bear greatly on the type of evaporator used and the pressure and temperature of the evaporation process. The factors to be considered in the operation include concentration in the liquid, foaming, solubility, scale deposition, and materials of construction. Figure 7.15 shows a typical single-effect evaporator.

In a single-effect evaporator, the latent heat of condensation of the steam is transferred through a heating surface to vaporize solvent from a boiling solution. Two enthalpy balances are needed: one for the steam side and one for the solution side. It is assumed that the superheat and the subcooling

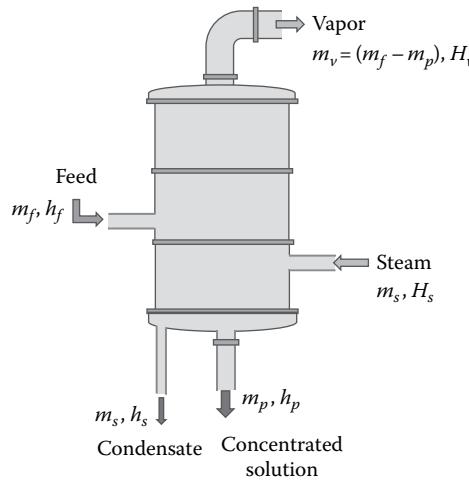


FIGURE 7.15 A single-effect evaporator.

of the condensate are negligible. The difference between the enthalpy of the steam and that of the condensate is represented by the latent heat of condensation of the steam:

$$q_s = m_s (h_s - H_s) = -m_s \lambda_s$$

where

h_s is the specific enthalpy of condensate

H_s is the specific enthalpy of steam

m_s is the flow rate of steam

q_s is the heat transfer rate through heating surface from steam

λ_s is the latent heat of condensation of steam

The enthalpy balance for the solution side yields

$$q = (m_f - m_p) H_v - m_f h_f + m_p h_p$$

where

h_f is the specific enthalpy of thin liquor

H_v is the specific enthalpy of vapor

h_p is the specific enthalpy of concentrated solution

q is the heat transfer rate from heating surface to the liquid solution

In the absence of heat losses, the heat transferred to the heating surface from steam equals that transferred from the heating surface to the liquid, and $q = -q_s$. Combination of the equations about these quantities gives

$$q = m_s \lambda_s = (m_f - m_p) H_v - m_f h_f + m_p h_p$$

h_f and h_p depend upon the temperature and concentration of the solution. From the enthalpy and mass balances, the heat transfer area A and the first effect m_{ev} are given by

$$A = \frac{q}{U(T_s - T_v)}, \quad m_{ev} = m_f - m_p$$

where T_s and T_v are temperature of saturated steam and saturated vapor, respectively.

The function *sinevap* calculates the single-effect evaporation process. The syntax is

```
res = sinevap(evdat)
```

The function requires the structure variable evdat, which consists of seven fields: evdat.mf is the feed flow rate (lb/hr), evdat.Tf is the feed temperature (°F), evdat.xf is the solid fraction in the feed solution, evdat.xp is the solid fraction in the concentrated solution, evdat.Ps is the inlet steam pressure (psia), evdat.Pv is the outlet vapor pressure (psia), and evdat.U is the overall heat transfer coefficient (Btu/(ft² hr°F)).

The function *sinevap* executes the following calculation procedure:

1. Converts the units of temperature and pressure to K and Pa, respectively
2. Finds the saturation temperatures of the steam and the solution using the equation

$$f(\tau) = a_1\tau + a_2\tau^{1.5} + a_3\tau^3 + a_4\tau^{3.5} + a_5\tau^4 + a_6\tau^{7.5} - (1-\tau)\ln\left(\frac{p}{p_c}\right) = 0$$

3. Uses the function *satsteam* to calculate properties at the saturation temperature. When this function is called, the unit of temperature should be converted to °C
4. Determines *q* and *A* from the mass balance

```
function res = sinevap(evdat)
% Calculation of single-effect evaporator
% Data
Tc = 647.096; % critical temperature of H2O (K)
Pc = 22064000; % critical pressure of H2O (Pa)
T0 = 273.15;
xf = evdat.xf; xp = evdat.xp; mf = evdat.mf;
Tf = (evdat.Tf-32)/1.8 + 273.15; % F->K
Ps = evdat.Ps*6894.757; Pv = evdat.Pv*6894.757; % psia->Pa
U = evdat.U;
cf = 1/2326; % J/kg->Btu/lb
% Saturation temperature
a = [-7.85951783 1.84408259 -11.7866497 22.6807411...
-15.9618719 1.80122502];
gs = @(x) a(1)*x + a(2)*x^1.5 + a(3)*x^3 + a(4)*x^3.5 + a(5)*x^4 +...
a(6)*x^7.5 - (1-x)*log(Ps/Pc);
gv = @(x) a(1)*x + a(2)*x^1.5 + a(3)*x^3 + a(4)*x^3.5 + a(5)*x^4 +...
a(6)*x^7.5 - (1-x)*log(Pv/Pc);
xs = fzero(gs,0.5); xv = fzero(gv,0.5);
Ts = Tc*(1 - xs); Tv = Tc*(1 - xv);
% Enthalpy
sts = satsteam(Ts-T0); stv = satsteam(Tv-T0); stf = satsteam(Tf-T0);
Hs = sts.hV*cf; hs = sts.hL*cf; % steam
Hv = stv.hV*cf; hv = stv.hL*cf; % solution
hf = stf.hL*cf;
% Material balance
mp = xf*mf/xp; % Btu/hr
mev = mf - mp;
q = mev*Hv - mf*hf + mp*hv;
ms = q/(Hs - hs);
Ts = (Ts - 273.15)*1.8+32; % K->F
Tv = (Tv - 273.15)*1.8+32;
dT = Ts - Tv;
A = q/U/dT;
```

Example 7.9: Single-Effect Evaporator

A vertical-tube single-effect evaporator is used to concentrate 29,000 lb/hr of a 25% solution of organic colloid to 60% solution. The feed temperature is 60°F. Assume that the boiling-point elevation is negligible, that physical properties of the solution are similar to those of water, and that the specific enthalpy of the solution, h_{pr} , is the same as that of the steam, h_v . The absolute pressure of the saturated steam being introduced is 25 psia. The absolute pressure in the vapor space is 1.69 psia and the overall heat transfer coefficient is 300 Btu/(ft² hr°F). Calculate the heating surface required (ft²) and the amount of steam consumed (lb/hr).

Solution

The following commands produce desired outputs:

```
>> evdat.mf = 29000; evdat.Tf = 60; evdat.xf = 0.25;
>> evdat.xp = 0.6; evdat.Ps = 25; evdat.Pv = 1.69; evdat.U = 300;
>> res = sinevap(evdat)
res =
    Ts: 240.0324
    Tv: 119.8916
    ms: 2.0039e+04
    hL: 87.8958
    hV: 1.1132e+03
    A: 529.3588
    q: 1.9079e+07
```

We can see that the amount of steam consumed is $m_s = 20,039$ lb/hr and the heating surface required is 529.3588 ft².

7.3.2 MULTIPLE-EFFECT EVAPORATORS

There are several methods of feeding in the evaporation process. The usual method of feeding a multiple-effect evaporator is to introduce the thin liquid into the first effect and send it in turn through the other effects. This method of feeding is called forward-feed. [Figure 7.16](#) shows a schematic of a forward-feed triple-effect evaporator. The fresh feed is added to the first effect and flows to the next in the same direction as the vapor flow. The concentration of the liquid increases from the first effect to the last effect. Connections are made so that the vapor from one effect serves as the heating medium for the next effect. [Figure 7.17](#) shows a typical multiple-effect evaporator.

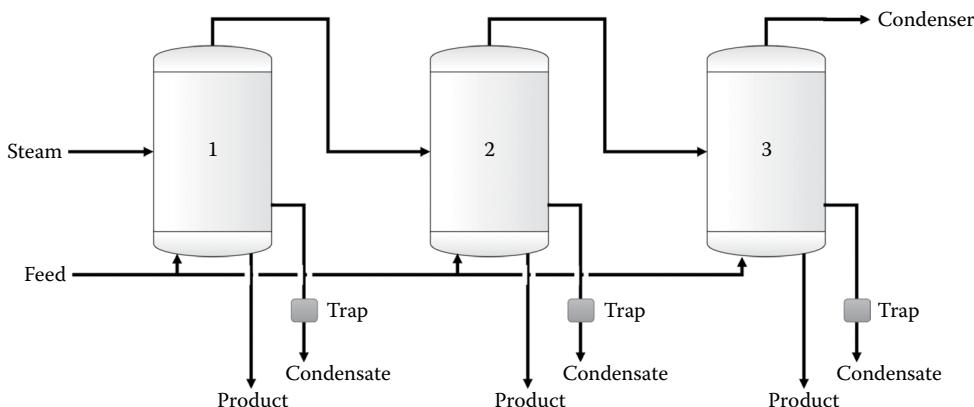


FIGURE 7.16 Schematic of a forward-feed triple-effect evaporator.

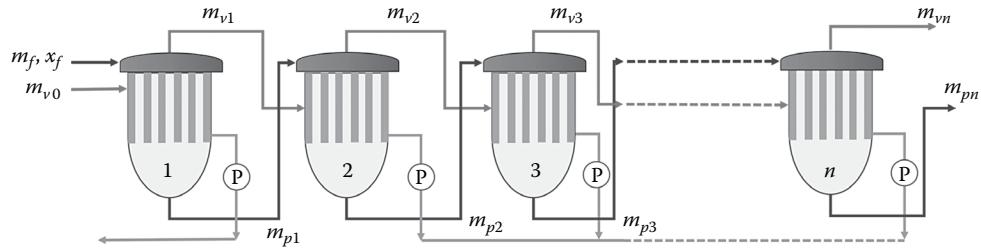


FIGURE 7.17 Simplified diagram of forward-feed m -effect evaporator.

Let the evaporation continue until the n th effect. Let the flow rate and the enthalpy of the liquor from the k th effect be m_{pk} and h_{pk} , respectively, and those of the vapor phase from the k th effect be m_{vk} and H_{vk} , respectively. From the overall material balance $m_f x_f = m_{pn} x_{pn}$, we have

$$m_{pn} = \frac{m_f x_f}{x_{pn}}$$

The mass and energy balances on the first effect yield

$$m_f = m_{p1} + m_{v1}, \quad m_f h_f + m_{v0} (H_{v0} - h_{v0}) = m_{v1} H_{v1} + m_{p1} h_{p1}$$

The mass and energy balances on the k th effect yield

$$m_{p,k-1} = m_{pk} + m_{vk}, \quad m_{p,k-1} h_{p,k-1} + m_{v,k-1} (H_{v,k-1} - h_{v,k-1}) = m_{pk} h_{pk} + m_{vk} H_{vk}$$

The flow rate of liquor from the k th effect is given by

$$m_{pk} = m_f - m_{v1} - m_{v2} - \cdots - m_{vk} = m_f - \sum_{i=1}^k m_v$$

Combination of these equations after rearrangement gives

$$\text{Effect 1: } (H_{v0} - h_{v0}) m_{v0} + (h_{p1} - H_{v1}) m_{v1} = (h_{p1} - h_f) m_f$$

$$\text{Effect 2: } (H_{v1} - h_{v1} - h_{p1} + h_{p2}) m_{v1} + (h_{p2} - H_{v2}) m_{v2} = (h_{p2} - h_{p1}) m_f$$

$$\begin{aligned} \text{Effect } k: & (h_{pk} - h_{p,k-1}) \sum_{i=1}^{k-2} m_{vi} + (H_{v,k-1} - h_{v,k-1} + h_{pk} - h_{p,k-1}) m_{v,k-1} \\ & + (h_{pk} - H_{vk}) m_{vk} = (h_{pk} - h_{p,k-1}) m_f \quad (3 \leq k \leq n-1) \end{aligned}$$

$$\begin{aligned} \text{Effect } n: & (H_{vn} - h_{p,n-1}) \sum_{i=1}^{n-2} m_{vi} + (H_{vn} + H_{v,n-1} - h_{v,n-1} - h_{p,n-1}) m_{v,n-1} \\ & = \left\{ (h_{pn} - h_{p,n-1}) + (H_{vn} - h_{pn}) \left(1 - \frac{x_f}{x_{pn}} \right) \right\} m_f \end{aligned}$$

The last equation can be obtained by using

$$m_{pn} = m_f - \sum_{i=1}^n m_{vi} = \frac{x_f}{x_{pn}} m_f, \quad m_{vn} = \left(1 - \frac{x_f}{x_{pn}}\right) m_f - \sum_{i=1}^{n-1} m_{vi}$$

The heat transfer area is considered to be the same in each effect: $A_1 = A_2 = \dots = A_n$. Then the temperature drop between adjacent effects is given by

$$\Delta T_i = \frac{\frac{1}{U_i}}{\frac{1}{U_1} + \frac{1}{U_2} + \dots + \frac{1}{U_n}} \cdot \Delta T_{total}$$

The heat transfer rate (q) at each effect can be calculated from the mass and energy balances. The heat transfer area (A) can be determined using the value of q . The calculation is repeated until the same heat transfer area is obtained for each effect. The calculation procedure can be summarized as follows:

1. Determine the saturation temperatures for steam and vapor at a given pressure. Evaluate the enthalpy at the saturation temperature.
2. Calculate ΔT_i and determine the saturation temperature in each effect, T_i .
3. Calculate the enthalpy of vapor and liquid phases at the temperature T_i .
4. Find the flow rate of vapor from each effect using mass and energy balances on each effect.
5. Determine the heat transfer rate in each effect and calculate the heat transfer area A_i .
6. Compare the values of A_i at each effect. If all A_i 's are the same each other, stop the procedure. If not, calculate $\Delta T_i = q_i/A_i U_i$ and go to step 3.

The functions *multievapSI* and *multievap* both perform the calculation procedure for the multiple-effect evaporation. The function *multievapSI* uses the SI unit system, while the function *multievap* employs the English unit system. The basic syntax is

```
res = multievapSI(evdat)
```

The structure *evdat* consists of seven fields. The function requires the structure variable *evdat*, which consists of seven fields: *evdat.xp* is the solid fraction in the concentrated solution at the final effect, *evdat.xf* is the solid fraction in the feed solution, *evdat.Ps* is the inlet steam pressure (Pa), *evdat.Pn* is the outlet vapor pressure from the last effect (Pa), *evdat.mf* is the feed flow rate (kg/hr), *evdat.Tf* is the feed temperature (°C), and *evdat.U* is the overall heat transfer coefficient at each effect (J/(m²·hr·K)). The output variable *res* is a structure containing the results and consists of four fields: *res.T* is a vector containing temperature at each effect (°C), *res.A* is a vector of the heat transfer area (m²), *res.mv* is a vector of vapor flow rate from each effect (kg/hr), and *res.iter* denotes the number of iteration.

```
function res = multievapSI(evdat)
% Multiple-effect evaporator calculation (SI unit system)
% Data
crit = 1e-3;
Tc = 647.096; % critical temperature of H2O (K)
Pc = 22064000; % critical pressure of H2O (Pa)
T0 = 273.15;
xf = evdat.xf; xp = evdat.xp; mf = evdat.mf;
```

```

Tf = evdat.Tf + T0; % C -> K
Ps = evdat.Ps; % steam pressure (Pa)
Pn = evdat.Pn; % pressure of the final effect (Pa)
U = evdat.U; % overall heat transfer coefficient
n = length(U);
% Saturation temperature
a = [-7.85951783 1.84408259 -11.7866497 22.6807411...
-15.9618719 1.80122502];
gs = @(x) a(1)*x + a(2)*x^1.5 + a(3)*x^3 + a(4)*x^3.5 + a(5)*x^4 +...
a(6)*x^7.5 - (1-x)*log(Ps/Pc);
gn = @(x) a(1)*x + a(2)*x^1.5 + a(3)*x^3 + a(4)*x^3.5 + a(5)*x^4 +...
a(6)*x^7.5 - (1-x)*log(Pn/Pc);
xs = fzero(gs,0.5); xn = fzero(gn,0.5);
Ts = Tc*(1 - xs); % steam temperature (K)
Tn = Tc*(1 - xn); % temperature of the final effect (K)
% Enthalpy
sts = satsteam(Ts-T0); % saturated steam
stv = satsteam(Tn-T0); % final effect
stf = satsteam(Tf-T0); % feed
Hv0 = sts.hV; hp0 = sts.hL; % J/kg
Hvn = stv.hV; hpn = stv.hL;
hf = stf.hL;
% Mass balances and temperature drop
mpn = xf*mf/xpn; % kg/hr
dTtotal = Ts - Tn; % K
sumU = sum(1./U);
dT = (1./U)*dTtotal/sumU; % K
critA = 10;
oldA = 10*ones(1,n);
iter = 0;
while critA >= crit
    T(1) = Ts - dT(1);
    for j = 2:n
        T(j) = T(j-1) - dT(j);
    end
    for j = 1:n
        sprop(j) = satsteam(T(j)-T0);
        Hv(j) = sprop(j).hV;
        hp(j) = sprop(j).hL;
    end
    % balance equations
    evM = [Hv0-hp0 hp(1)-Hv(1) 0;...
            0 Hv(1)+hp(2)-2*hp(1) hp(2)-Hv(2) ;...
            0 Hv(3)-hp(2) Hv(2)+Hv(3)-2*hp(2) ];
    evb = [hp(1)-hf; hp(2)-hp(1); Hv(3)-hp(2)+(hp(3)-Hv(3))*xf/xpn]*mf;
    mv = evM\evb;
    q(1) = mv(1)*(Hv0-hp0);
    for j = 2:n
        q(j) = mv(j)*(Hv(j)-hp(j));
    end
    Area = q./(U.*dT);
    avgA = sum(Area)/n;
    critA = sum(abs(Area - oldA));
    dT = dT.*Area/avgA;
    if abs(sum(dT) - dTtotal) >= crit
        dT = dT*dTtotal/sum(dT);
    end
end

```

```

    end
    iter = iter + 1;
    oldA = Area;
end
% Results
res.T = T-T0; % vector of temperature of each effect (C)
res.A = Area; % vector of area of each effect (m^2)
res.mv = mv; % vector of vapor flow rate from each effect (kg/hr)
res.iter = iter; % number of iterations
end

```

Example 7.10: Multiple-Effect Evaporator

A triple-effect forward-feed evaporator is being used to evaporate an organic colloid solution containing 15% solids to a concentrated solution of 60%. Saturated steam at 205,603 Pa is being used, and the pressure in the vapor phase of the third effect is 8,756 Pa. The feed rate is 20,412 kg/hr at 15.6°C. The heat capacity of the liquid solutions is the same as that of water at the whole concentration range. The coefficients of heat transfer have been estimated as $U_1 = 1.084 \times 10^7$, $U_2 = 7.155 \times 10^6$, and $U_3 = 3.986 \times 10^6$ J/(m²·hr·K). If the heat transfer areas of each of the three effects are to be equal, calculate the heat transfer area (m²) and the amount of steam consumed (kg/hr). The boiling-point elevation of the solutions is assumed to be negligible.

Solution

The enthalpy balance on each effect gives

$$\text{Effect 1: } (H_{v0} - h_{v0})m_{v0} + (h_{p1} - H_{v1})m_{v1} = (h_{p1} - h_f)m_f$$

$$\text{Effect 2: } (H_{v1} - h_{v1} - h_{p1} + h_{p2})m_{v1} + (h_{p2} - H_{v2})m_{v2} = (h_{p2} - h_{p1})m_f$$

$$\text{Effect 3: } (H_{v3} - h_{p2})m_{v1} + (H_{v3} + H_{v2} - h_{v2} - h_{p2})m_{v2} = \left\{ (H_{v3} - h_{p2}) + (h_{p3} - H_{v3}) \frac{x_f}{x_{p3}} \right\} m_f$$

Assuming that the enthalpy of the condensate is the same as that of the solution (i.e., $h_{vi} = h_{pi}$), we have

$$(H_{v0} - h_{p0})m_{v0} + (h_{p1} - H_{v1})m_{v1} = (h_{p1} - h_f)m_f$$

$$(H_{v1} - 2h_{p1} + h_{p2})m_{v1} + (h_{p2} - H_{v2})m_{v2} = (h_{p2} - h_{p1})m_f$$

$$(H_{v3} - h_{p2})m_{v1} + (H_{v3} + H_{v2} - 2h_{p2})m_{v2} = \left\{ (H_{v3} - h_{p2}) + (h_{p3} - H_{v3}) \frac{x_f}{x_{p3}} \right\} m_f$$

Rearrangement of these equations yields

$$\begin{bmatrix} H_{v0} - h_{p0} & h_{p1} - H_{v1} & 0 \\ 0 & H_{v1} - 2h_{p1} + h_{p2} & h_{p2} - H_{v2} \\ 0 & H_{v3} - h_{p2} & H_{v3} + H_{v2} - 2h_{p2} \end{bmatrix} \begin{bmatrix} m_{v0} \\ m_{v1} \\ m_{v2} \end{bmatrix} = \begin{bmatrix} (h_{p1} - h_f) \\ (h_{p2} - h_{p1}) \\ (H_{v3} - h_{p2}) + (h_{p3} - H_{v3}) \frac{x_f}{x_{p3}} \end{bmatrix} m_f$$

The following commands show the results of calculations by the function *multievapSI*:

```

>> evdat.xp = 0.60; evdat.xf = 0.15; evdat.Ps = 205602.9; evdat.Pn = 8756;
>> evdat.mf = 20412; evdat.Tf = 15.6; evdat.U = [10.834 7.155 3.986]*10^6;
>> results = multievapSI(evdat)
results =
    T: [100.4974 81.7076 43.2312]
    A: [79.4099 79.4099 79.4099]
    mv: [3x1 double]
    iter: 6
>> results.mv
ans =
    1.0e+03 *
    8.0540
    4.6341
    5.0781

```

We can see that the heat transfer area was converged to 79.41 m² after six iterations.

7.4 ABSORPTION

In the absorption process, a soluble vapor is absorbed by means of a liquid in which the solute gas is more or less soluble from its mixture with an inert gas. Usually, absorption is used to remove impurities, contaminants, and pollutants or to recover valuable chemicals. Figure 7.18 shows a typical gas absorption process.

For the absorber shown in Figure 7.18, L and V are the molar flow rates of liquid and gas phases, respectively, and x_i and y_i are the mole fractions of liquid and gas phases at the i th stage, respectively. Assuming constant L and V and thermodynamic equilibrium in each stage, the mass balance on the i th stage gives

$$\frac{d(Mx_i + Wy_i)}{dt} = Lx_{i-1} + Vy_{i+1} - Lx_i - Vy$$

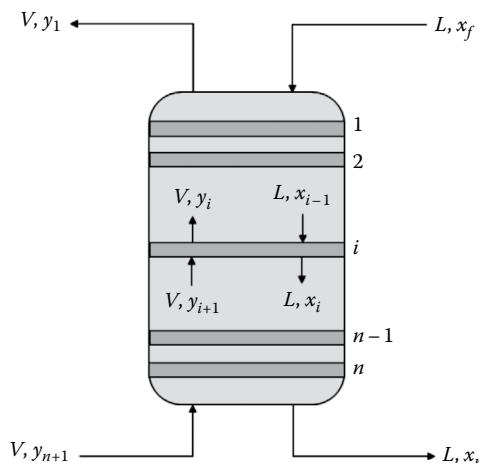


FIGURE 7.18 Typical gas absorption process.

where M and W are the molar holdups of liquid and gas phases for the i th stage, respectively. W may be negligible compared to M , and M is assumed to be constant for each stage. The mass balance can be rewritten as

$$\frac{dx_i}{dt} = \frac{L}{M} x_{i-1} + \frac{V}{M} y_{i+1} - \frac{L}{M} x_i - \frac{V}{M} y_i$$

The equilibrium relation in the gas–liquid may be represented as $y_i = ax_i$. Substitution of this equilibrium relation into the mass balance equation gives

$$\frac{dx_i}{dt} = \frac{L}{M} x_{i-1} - \frac{(L+Va)}{M} x_i + \frac{Va}{M} x_{i+1}$$

$$\frac{dx_1}{dt} = \frac{L}{M} x_f - \frac{(L+Va)}{M} x_1 + \frac{Va}{M} x_2$$

$$\frac{dx_n}{dt} = \frac{L}{M} x_{n-1} - \frac{(L+Va)}{M} x_n + \frac{V}{M} y_{n+1}$$

For $n = 5$, these equations can be represented as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{bmatrix} = \begin{bmatrix} -\frac{(L+Va)}{M} & \frac{Va}{M} & 0 & 0 & 0 \\ \frac{L}{M} & -\frac{(L+Va)}{M} & \frac{Va}{M} & 0 & 0 \\ 0 & \frac{L}{M} & -\frac{(L+Va)}{M} & \frac{Va}{M} & 0 \\ 0 & 0 & \frac{L}{M} & -\frac{(L+Va)}{M} & \frac{Va}{M} \\ 0 & 0 & 0 & \frac{L}{M} & -\frac{(L+Va)}{M} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} \frac{L}{M} & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{V}{M} \end{bmatrix} \begin{bmatrix} x_f \\ y_6 \end{bmatrix}$$

or

$$\dot{x} = Ax + Bu$$

At steady state, the time derivatives are zero:

$$0 = Ax_s + Bu_s$$

Thus, the steady-state value of x is given by

$$x_s = -A^{-1}Bu_s$$

Introduction of a step change in feed composition, Δu , yields

$$\Delta\dot{x} = A\Delta x + B\Delta u, \quad \Delta y = C\Delta x + D\Delta u = \begin{bmatrix} \Delta x_5 \\ \Delta y_1 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ a & 0 & 0 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

where

Δy is a response vector

Δx_5 and Δy_1 are the mole fractions of liquid and gas phases from the absorption column

Example 7.11: Benzene Absorption Column

A five-stage absorption column is to be used to remove benzene contained in the process gas using an oil stream. The oil feed flow rate is 1.33 kg mol/min, and the gas feed rate and the mole fraction of benzene in the gas feed are 1.667 kg mol air/min and 0.1, respectively. The liquid molar holdup for each stage is 6.667 kg mol and the equilibrium relation is given by $y_i = 0.5x_i$. The oil feed does not contain any benzene ($x_f = 0$).

1. Calculate the steady-state composition for each stage.
2. The benzene composition of the gas stream entering the column (y_6) suddenly changed from 0.1 to 0.15. Plot the compositions of the liquid and vapor streams leaving the column as a function of time. Also plot the profile of the liquid-phase composition for each stage.

Solution

The numerical data yield the following matrices:

$$A = \begin{bmatrix} -0.325 & 0.125 & 0 & 0 & 0 \\ 0.2 & -0.325 & 0.125 & 0 & 0 \\ 0 & 0.2 & -0.325 & 0.125 & 0 \\ 0 & 0 & 0.2 & -0.325 & 0.125 \\ 0 & 0 & 0 & 0.2 & -0.325 \end{bmatrix}, \quad B = \begin{bmatrix} 0.2 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0.25 \end{bmatrix}, \quad U_s = \begin{bmatrix} x_{fs} \\ y_{6s} \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.1 \end{bmatrix}$$

$$x_s = -A^{-1}BU_s$$

1.

```
>> A = [-0.325 0.125 0 0 0;0.2 -0.325 0.125 0 0;0 0.2 -0.325 0.125 0;
0 0.2 -0.325 0.125;0 0 0.2 -0.325];
>> B = [0.2 0;0 0;0 0;0 0.25]; Us = [0.0;0.1];
>> xs = -inv(A)*B*Us
xs =
0.0076
0.0198
0.0392
0.0704
0.1202
```

Thus,

$$x_s = -A^{-1}BU_s = \begin{bmatrix} 0.0076 \\ 0.0198 \\ 0.0392 \\ 0.0794 \\ 0.1202 \end{bmatrix}$$

2. The step change is $\Delta u = \begin{bmatrix} 0 \\ 0.05 \end{bmatrix}$, which is equal to the unit step change multiplied by 0.05. Addition of the steady-state values to the step response yields the actual composition. The script *absbz* generates desired profile plots shown in Figures 7.19 and 7.20.

```
% absbz.m
A = [-0.325 0.125 0 0 0;0.2 -0.325 0.125 0 0;0 0.2 -0.325 0.125 0;
0 0.2 -0.325 0.125;0 0 0.2 -0.325];
B = [0.2 0;0 0;0 0;0 0.25]; C = [0 0 0 1;0.5 0 0 0 0]; D = [0 0;0 0];
```

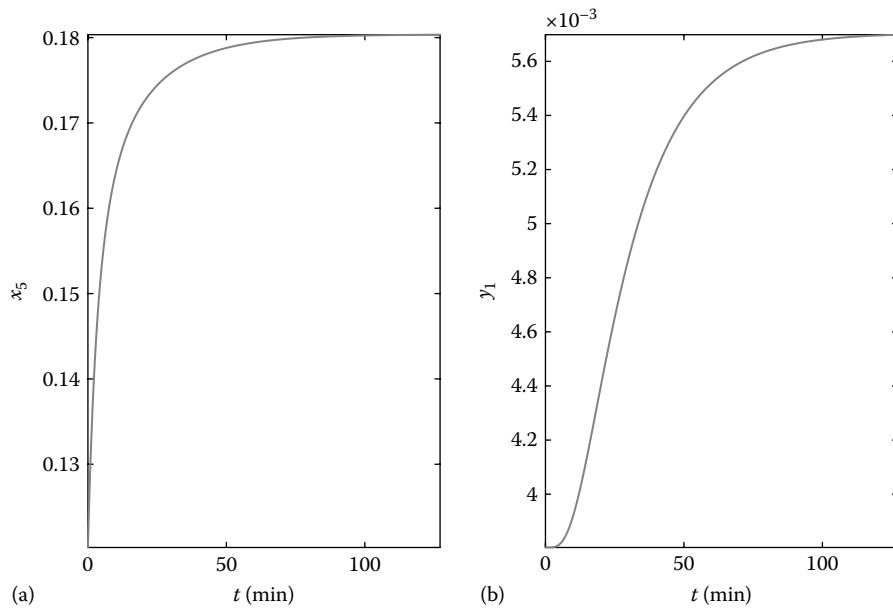


FIGURE 7.19 Composition profiles of outlet stream.

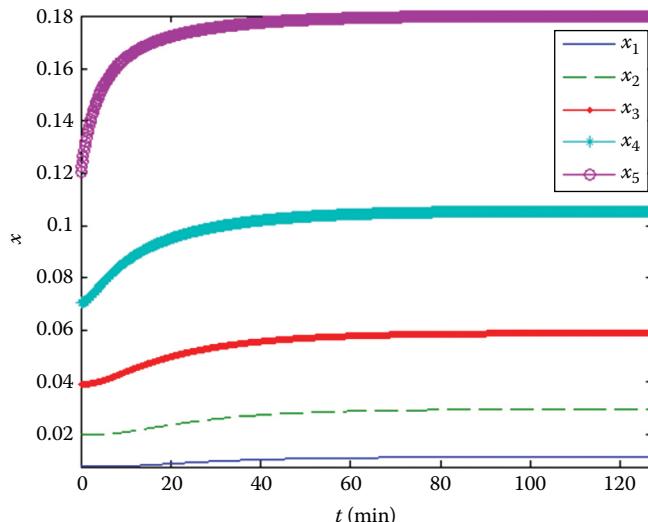


FIGURE 7.20 Liquid-phase mole fractions in each stage versus time.

```

Us = [0.0;0.1]; xs = -inv(A)*B*Us; ys = C*xs + D*Us; % Steady-state
[y,x,t] = step(A,B,C,D,2); y = 0.05*y; x = 0.05*x;
for k = 1:2, y(:,k) = y(:,k) + ys(k); end; % Y(t) = ys + y(t)
for k = 1:5, x(:,k) = x(:,k) + xs(k); end; % X(t) = xs + x(t)
figure(1)
subplot(1, 2), plot(t,y(:,1)), xlabel('t(min)'), ylabel('x_5'), axis tight
subplot(1, 2), plot(t,y(:,2)), xlabel('t(min)'), ylabel('y_1'), axis tight
figure(2)
plot(t,x(:,1),t,x(:,2),'--',t,x(:,3),'-',t,x(:,4),'*-',t,x(:,5),'o-')
xlabel('t(min)'), ylabel('x'), axis tight
legend('x_1','x_2','x_3','x_4','x_5')

```

7.5 BINARY DISTILLATION

7.5.1 McCABE/THIELE METHOD²³

Figure 7.21 shows a simple distillation column with a total condenser and a reboiler. The feed typically enters close to the middle of the column. Vapor flows from stage to stage up the column while liquid flows from stage to stage down the column. The vapor from the top tray is condensed to liquid in the condenser and a portion of that liquid, L , is returned as reflux. The rest of that vapor is withdrawn as the overhead product stream, D . The reflux ratio is defined as

$$R = \frac{L}{D}$$

A portion of the liquid at the bottom of the column is withdrawn as a bottoms product while the rest is vaporized in the reboiler and returned to the column.

The operating line for the rectifying section (top section of the column, above the feed stage n_F) can be represented as

$$y = \left(\frac{R}{R+1} \right) x + \left(\frac{1}{R+1} \right) x_D$$

where

x and y are the mole fractions of key (light key) component in the liquid and the vapor, respectively
 x_D is the mole fraction of key component in the distillate

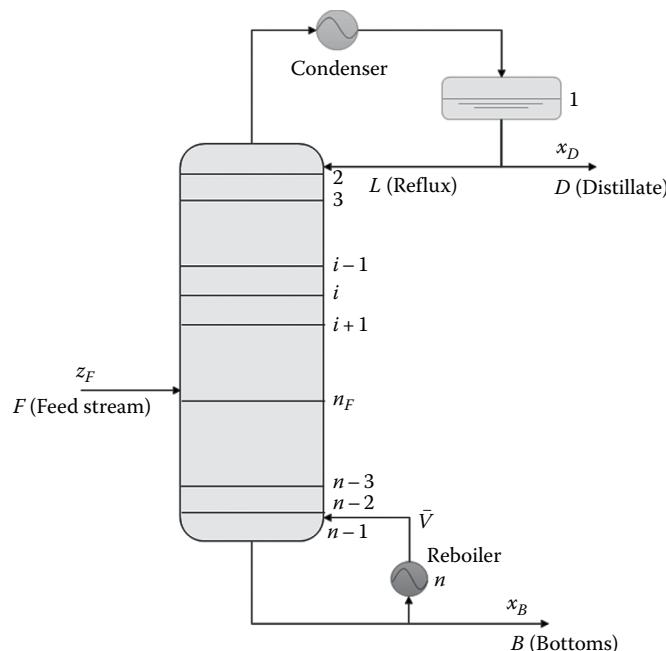


FIGURE 7.21 Schematic diagram of a distillation column. (From Henley, E.J. et al., *Separation Process Principles*, 3rd ed., John Wiley & Sons, Inc., Hoboken, NJ, 2011, pp. 285–290.)

The operating line for the stripping section (bottom section of the column, below the feed stage) can be expressed as

$$y = \left(\frac{V_B + 1}{V_B} \right) x - \left(\frac{1}{V_B} \right) x_B$$

where

V_B , the buildup ratio, is the ratio of the flow rate of vapor leaving the reboiler

\bar{V} , to the bottoms product rate, B , and is given by $V_B = \bar{V}/B$

x_B is the mole fraction of key component in the bottoms product

The condition of the feed stream fed into the feed stage can be characterized by the parameter q , which is defined as the ratio of the increase in molar reflux rate across the feed stage to the molar feed rate:

$$q = \frac{\bar{L} - L}{F}$$

Alternatively, q can be obtained from the mass balance around the feed stage as

$$q = 1 + \frac{\bar{V} - V}{F}$$

For subcooled liquid feed $q > 1$, for bubble-point liquid feed $q = 1$, for partially vaporized feed $0 < q < 1$, for dew-point vapor feed $q = 0$, and for superheated vapor feed $q < 0$. q -line equation is represented as

$$y = \left(\frac{q}{q-1} \right) x - \left(\frac{z_F}{q-1} \right)$$

From the rectifying operating line and the q -line equation, we have

$$x_q = \frac{(R+1)z_F + (q-1)x_D}{R+q}, \quad y_q = \frac{Rz_F + qx_D}{R+q}$$

If these relations are substituted to the stripping operating line to eliminate V_B , the stripping operating line can be rewritten as

$$y = \left(\frac{y_q - x_B}{x_q - x_B} \right) x - \left(\frac{y_q - x_q}{x_q - x_B} \right) x_B$$

The relative volatility, α , represents a numerical measure of separation. For a binary mixture, the relative volatility can be expressed in terms of equilibrium vapor and liquid mole fractions from the K -value:

$$\alpha = \frac{y/x}{(1-y)/(1-x)}$$

Solving this equation for y ,

$$y = \frac{\alpha x}{1 + (\alpha - 1)x}$$

The function *bindistMT* determines the number of stages and the operating lines for a binary distillation using the McCabe/Thiele method. The basic syntax is

```
[feedn, totaln] = bindistMT(alpha, q, zf, xd, xb, R)
```

where α is the relative volatility, q is the feed condition parameter, zf is the mole fraction of key component in the feed stream, xd is the mole fraction of key component in the distillate, xb is the mole fraction of key component in the bottoms product, and R is the reflux ratio. In the output vector, $feedn$ is the location of the feed stage and $totaln$ is the number of total stages.

```
function [feedn, totaln] = bindistMT(alpha, q, zf, xd, xb, R)
% Calculation of binary distillation using McCabe/Thiele method
% input
% alpha: relative volatility
% q: feed condition parameter
% zf,xd,xb: mole fractions of feed, distillate and bottoms
% R: reflux ratio
% output
% feedn: feed stage
% totaln: number of total stages
% Initialization and calculation of equilibrium curve
y = 0:0.1:1; ye = y; xe = ye./(alpha + (1-alpha)*ye);
xq = ((R+1)*zf + (q-1)*xd)/(R + q); yq = (R*zf + q*xd)/(R + q);
figure(1)
plot(xe,ye,'r'); hold on, axis([0 1 0 1]);
set(line([0 1],[0 1]),'Color',[0 0 0]);
set(line([xd xq],[xd yq]),'Color',[1 0 1]);
set(line([zf xq],[zf yq]),'Color',[1 0 1]);
set(line([xb xq],[xb yq]),'Color',[1 0 1]);
%Rectifying section
i = 1; xop(1) = xd; yop(1) = xd; y = xd;
while (xop(i) > xq)
    xop(i+1) = y./ (alpha + (1-alpha)*y);
    yop(i+1)=R*xop(i+1)/(R+1) + xd/(R+1); % rectifying operating line
    y = yop(i+1);
    set(line([xop(i) xop(i+1)], [yop(i) yop(i)]), 'Color', [0 0 1]);
    if (xop(i+1) > xq)
        set(line([xop(i+1) xop(i+1)], [yop(i) yop(i+1)]), 'Color', [0 0 1]);
    end
    i = i+1;
end
feedn = i-1;
% Stripping section
c1 = (yq - xb)/(xq - xb); c2 = (yq - xq)/(xq - xb);
yop(i) = c1*xop(i) - c2*xb; y = yop(i);
set(line([xop(i) xop(i)], [yop(i-1) yop(i)]), 'Color', [0 0 1]);
while (xop(i)>xb)
    xop(i+1) = y./ (alpha + (1-alpha)*y);
    yop(i+1) = c1*xop(i+1) - c2*xb; y = yop(i+1);
    set(line([xop(i) xop(i+1)], [yop(i) yop(i)]), 'Color', [0 0 1]);
```

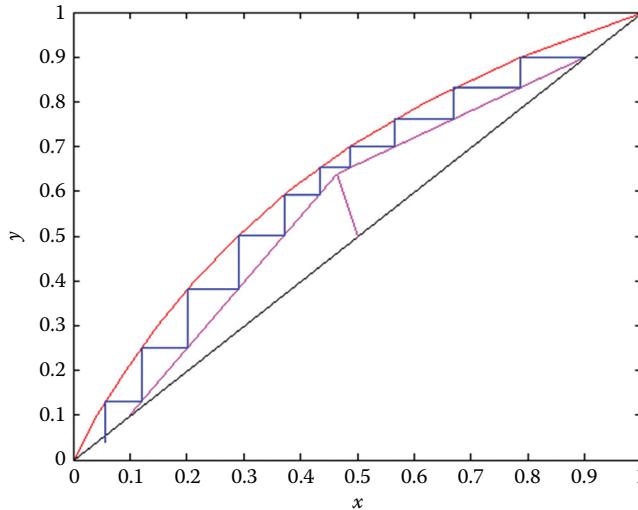


FIGURE 7.22 Determination of the number of stages by the McCabe-Thiele method.

```

if (xop(i+1) > xb)
    set(line([xop(i+1) xop(i+1)], [yop(i) yop(i+1)]), 'Color', [0 0 1]);
end
i=i+1;
end
set(line([xop(i) xop(i)], [yop(i-1) yop(i)]), 'Color', [0 0 1]);
hold off, xlabel('x'), ylabel('y'), totaln = i-1;
fprintf('Feed stage = %g\n', feedn);
fprintf('Number of stages = %g\n', totaln);

```

Example 7.12: McCabe-Thiele Operating Lines

A binary mixture is to be distilled in a distillation column to give a distillate of $x_D = 0.9$ and a bottoms composition of $x_B = 0.1$. The feed composition is $z_F = 0.5$ and the reflux ratio is $R = 1.5$. The feed is partially vaporized and $q = 0.8$. The equilibrium equation is given by $y = \alpha x / (1 + (1 - \alpha)x)$ with $\alpha = 2.45$. Determine the location of the feed stage and the number of total stages, and plot the equilibrium curve, q -line, and the operating lines as a function of liquid-phase mole fraction.

Solution

The following commands produce desired outputs and the plot shown in Figure 7.22:

```

>> alpha = 2.45; q = 0.80; zf = 0.5; xd = 0.9; xb = 0.1; R = 1.5;
>> [feedn, totaln] = bindistMT(alpha, q, zf, xd, xb, R);
Feed stage = 5
Number of stages = 10

```

7.5.2 IDEAL BINARY DISTILLATION

Figure 7.23 shows the feed stage and i th stage for a binary distillation column. The mass balance around the i th stage gives

$$\frac{d(M_i x_i)}{dt} = L_{i-1} x_{i-1} + V_{i+1} y_{i+1} - L_i x_i - V_i y_i$$

where M_i is the molar liquid holdup in the i th stage.

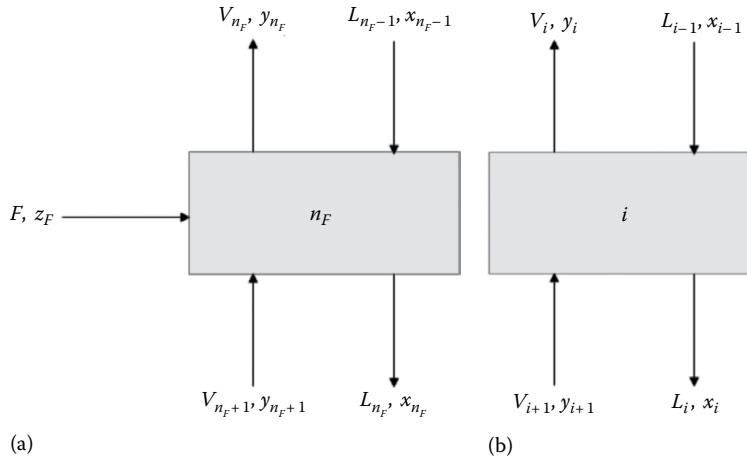


FIGURE 7.23 (a) Feed stage and (b) i th stage for a binary distillation column.

The vapor–liquid equilibrium curve is given by

$$y = \frac{\alpha x}{1 + (\alpha - 1)x}$$

and the molar flow rates of vapor and liquid are assumed to be constant. The molar liquid holdup, M_T , is constant for all stages. Overall mass balances for the feed stage, the condenser, and the reboiler are as follows:

$$L_{n_F} = L_{n_F-1} + F, \quad V_2 = L + D, \quad B = L_n - \bar{V}$$

The liquid flow rates for the rectifying section and the stripping section are given by

$$L_R = L, \quad L_S = L_R + Fq$$

and the vapor flow rates for the rectifying section and the stripping section are given by

$$V_S = \bar{V}, \quad V_R = V_s + F(1 - q)$$

Component balances are as follows:

$$\text{Condenser } (i=1): \frac{dx_1}{dt} = \frac{1}{M_D} [V_R (y_2 - x_1)]$$

$$\text{Rectifying section: } (i=2, \dots, n_F-1): \frac{dx_i}{dt} = \frac{1}{M_T} (L_R x_{i-1} + V_R y_{i+1} - L_R x_i - V_R y_i)$$

$$\text{Feed stage } (i=n_F): \frac{dx_{n_F}}{dt} = \frac{1}{M_T} (L_R x_{n_F-1} + V_S y_{n_F+1} + F z_F - L_S x_{n_F} - V_R y_{n_F})$$

$$\text{Stripping section } (i=n_F+1, \dots, n-1): \frac{dx_i}{dt} = \frac{1}{M_T} (L_S x_{i-1} + V_S y_{i+1} - L_S x_i - V_S y_i)$$

$$\text{Reboiler } (i = n): \frac{dx_n}{dt} = \frac{1}{M_B} (L_S x_{n-1} - B x_n - V_S y_n)$$

The MATLAB function *dyndist* defines the system of differential equations representing the dynamics of a binary distillation process. The basic syntax is

```
dx = dyndist(t, x, dpar, dels)
```

The input argument *dpar* is a structure variable consisting of parameter fields: *dpar.alpha* = α (relative volatility), *dpar.n* = n (number of total stages), *dpar.nf* = n_F (feed stage), *dpar.F* = F (feed flow rate), *dpar.zf* = z_F (feed composition), *dpar.q* = q (feed condition), *dpar.R* = R (reflux rate), *dpar.Vs* = V_s (reboiler vapor rate), *dpar.md* = M_D (condenser holdup), *dpar.mb* = M_B (reboiler holdup), and *dpar.mt* = M_T (liquid holdup in each stage). The input argument *dels* is a structure variable that defines step changes in operating variables.

```
function dx = dyndist(t, x, dpar, dels)
% System of differential equations for a binary distillation column
% Equilibrium relation: y=alpha*x/(1+(alpha-1)*x)
% Solver: [t x] = ode45(@dyndist, [t0 tf], x0, [], dpar, dels)
% Parameters
alpha = dpar.alpha; n = dpar.n; nf = dpar.nf; Fi = dpar.F;
zfi = dpar.zf; q = dpar.q; Ri = dpar.R; Vsi = dpar.Vs;
md = dpar.md; mb = dpar.mb; mt = dpar.mt;
delR = dels.delR; delRt = dels.delRt; delV = dels.delV; delVt = dels.delVt;
delz = dels.delz; delzt = dels.delzt; delF = dels.delF; delFt = dels.delFt;
% Changes in operating conditions
if t < delRt, R = Ri; else R = Ri + delR; end
if t < delVt, Vs = Vsi; else Vs = Vsi + delV; end
if t < delzt, zf = zfi; else zf = zfi + delz; end
if t < delFt, F = Fi; else F = Fi + delF; end
% Floe rates
Lr = R; Ls = R + F*q; B = Ls - Vs; D = F - B;
Vr = Vs + F*(1-q);
% Initialization and phase equilibrium
dx = zeros(n,1);
y = alpha*x./(1 + (alpha-1)*x);
% Condenser
dx(1) = Vr*(y(2)-x(1))/md;
% Rectifying section
for i = 2:nf-1;
dx(i) = (Lr*x(i-1)+Vr*y(i+1)-Lr*x(i)-Vr*y(i))/mt;
end
% Feed stage
dx(nf) = (Lr*x(nf-1)+Vs*y(nf+1)+F*zf-Ls*x(nf)-Vr*y(nf))/mt;
% Stripping section
for i = nf+1:n-1;
dx(i) = (Ls*x(i-1)+Vs*y(i+1)-Ls*x(i)-Vs*y(i))/mt;
end
% Reboiler
dx(n) = (Ls*x(n-1)-B*x(n)-Vs*y(n))/mb;
end
```

The composition profile may be sensitive to the initial conditions. It is common practice to use the steady-state values as the initial conditions. At steady state, the compositions do not change with

time and the derivative terms in the balance equations all become zero. The function *ssdist* calculates the steady-state liquid compositions for all stages. The syntax is

```
f = ssdist(x,dpar)
```

where *dpar* is the parameter structure whose fields are *dpar.alpha* = α , *dpar.n* = n , *dpar.nf* = n_f , *dpar.F* = F , *dpar.zf* = z_F , *dpar.q* = q , *dpar.R* = R , and *dpar.D* = D .

```
function f = ssdist(x,dpar)
alpha = dpar.alpha; n = dpar.n; nf = dpar.nf; F = dpar.F;
zf = dpar.zf; q = dpar.q; R = dpar.R; D = dpar.D;
Lr = R; B = F - D; Ls = R + F*q; Vs = Ls - B;
Vr = Vs + F*(1-q);
y = alpha*x./(1 + (alpha-1)*x);
f(1) = (Vr*y(2) - (D + R)*x(1)); % condenser
for i = 2:nf-1;
f(i) = Lr*x(i-1) + Vr*y(i+1) - Lr*x(i) - Vr*y(i);
end
f(nf) = Lr*x(nf-1) + Vs*y(nf+1) - Ls*x(nf) - Vr*y(nf) + F*zf;
for i = nf+1:n-1;
f(i) = Ls*x(i-1) + Vs*y(i+1) - Ls*x(i) - Vs*y(i);
end
f(n) = (Ls*x(n-1) - B*x(n) - Vs*y(n)); % reboiler
```

The system of nonlinear equations defined by the function *ssdist* can be solved using the built-in solver *fsolve*. The results obtained from the solution are used as initial values for the compositions.

Example 7.13: Dynamics of a Binary Distillation Column

A 30-stage column with the overhead condenser as stage 1, the feed stage as stage 15, and the reboiler as stage 30 is used to distil a binary mixture. The relative volatility, α , is 1.5. The feed rate is $F = 1$ mol/min, the feed composition is $z_F = 0.5$, and the feed condition is $q = 1$ (bubble-point liquid). The reflux flow rate is $R = 2.7$ mol/min and the vapor rate leaving the reboiler is 3.2 mol/min. The holdups in the condenser and the reboiler are both 5 mol, and the holdup in each stage is maintained constant as 0.5 mol.

At $t = 10$ min, there is 1% step change in the reflux flow rate. Plot the liquid compositions of the distillate and the bottoms for $0 \leq t \leq 400$. Also plot the liquid composition profile along the stages at $t = 400$ min.

Solution

The script *usedyndist* sets the field values of the structures *dpar* and *dels* for the given operating conditions. Then the function *ssdist* is called to calculate the steady-state compositions, which are used as initial values. Finally, the function *dyndist* is called and the built-in solver *ode45* is employed to solve the system of differential equations.

```
% usedyndist.m
dpar.alpha = 1.5; dpar.n = 30; dpar.nf = 15; dpar.F = 1;
dpar.zf = 0.5; dpar.q = 1; dpar.R = 2.7; dpar.Vs = 3.2;
dpar.D = 0.5; dpar.md = 5; dpar.mb = 5; dpar.mt = 0.5;
dels.delR = 0.01*dpar.R; dels.delRt = 10; dels.delV = 0; dels.delVt = 0;
dels.delz = 0; dels.delzt = 0; dels.delF = 0; dels.delFt = 0;
t0 = 0; tf = 400; x0 = 0.5*ones(1,dpar.n); nv = 1:dpar.n;
x0 = fsolve(@ssdist,x0,[],dpar); % steady-state to be used as initial conditions
for i = 1:length(x0), if x0(i) <= 0, x0(i) = -x0(i); end; end
[t x] = ode45(@dyndist,[t0 tf],x0,[],dpar,dels);
subplot(1, 2)
```

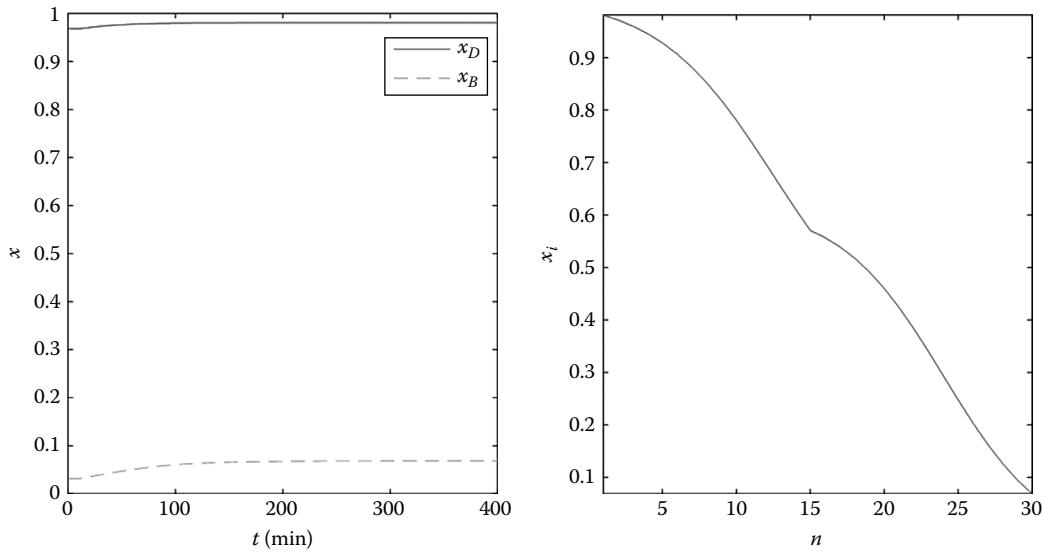


FIGURE 7.24 Composition profiles.

```
plot(t,x(:,1),t,x(:,end),'--'), xlabel('t(min)'), ylabel('x'), legend('x_D','x_B')
subplot(1, 2)
plot(nv,x(end,:)), xlabel('n'), ylabel('x_i'), axis tight
```

In the script, “dels.delR = 0.01*dpar.R;” denotes 1% step change in the reflux flow at “dels.delRt = 10;” (i.e., $t = 10$ min). The script *usedyndist* generates the composition profiles shown in Figure 7.24:

```
>> usedyndist
```

7.6 SHORTCUT CALCULATION METHOD FOR MULTICOMPONENT DISTILLATION

7.6.1 FENSKE-UNDERWOOD-GILLILAND METHOD

7.6.1.1 Fenske Equation: The Minimum Number of Stages

The minimum number of equilibrium stages, N_{\min} , can be calculated by using the Fenske equation:

$$N_{\min} = \frac{\log \left[\left(\frac{x_{LK}}{x_{HK}} \right)_D \left(\frac{x_{HK}}{x_{LK}} \right)_B \right]}{\log \left[\alpha_{(LK/HK)_{avg}} \right]}$$

where

N_{\min} is the minimum number of equilibrium stages including the reboiler and the condenser

x_{LK} is the mole fraction of the light key component

x_{HK} is the mole fraction of the heavy key component

$\alpha_j = k_j/k_{HK}$ is the relative volatility of the component j

$\alpha_{(LK/HK)_{avg}}$ is the geometric average volatility of the light key component given by $\alpha_{(LK/HK)_{avg}} = \sqrt{\alpha_{TD}\alpha_{TB}}$

In the calculation of $\alpha_{(LK/HK)_{avg}}$, the temperature at the top of the column, T_D (dew-point temperature of the distillate), and the temperature at the bottom of the column, T_B (bubble-point temperature of the bottoms), are determined, followed by the calculation of α_{TD} at T_D and α_{TB} at T_B .

If we use the component molar flow rates d and b , the minimum number of stages is given by

$$N_{\min} = \frac{\log \left[\left(\frac{d_{LK}}{d_{HK}} \right) \left(\frac{b_{HK}}{b_{LK}} \right) \right]}{\log \sqrt{\left(\alpha_{i,j} \right)_D \left(\alpha_{i,j} \right)_B}}$$

7.6.1.2 Underwood Equation: The Minimum Reflux

The Underwood equation is used to calculate the minimum reflux. First, the following equation is solved for the parameter θ :

$$1 - q = \sum_{j=1}^{n_c} \frac{\alpha_{jF} x_{jF}}{(\alpha_{jF} - \theta)}$$

where

q is the feed condition parameter

x_{jF} is the mole fraction of component j in the feed stream

α_{jF} is the relative volatility of component j in the feed

n_c is the total number of components

The minimum reflux ratio, R_{\min} , is calculated from the equation

$$R_{\min} + 1 = \sum_{j=1}^{n_c} \frac{\alpha_{jD} x_{jD}}{(\alpha_{jD} - \theta)}$$

where

α_{jD} is the relative volatility of component j in the distillate

x_{jD} is the mole fraction of component j in the distillate

$\alpha_{j,HK}$ may be used instead of α_{jF} and α_{jD} .

7.6.1.3 Gilliland Correlation²⁴

The number of theoretical equilibrium stages required for a given separation at a given reflux ratio is often determined by empirical correlations in terms of the abscissa, X , and the ordinate, Y :

$$X = \frac{R - R_{\min}}{R + 1}, \quad Y = \frac{N - N_{\min}}{N + 1}$$

The Gilliland correlation is an experimental curve that gives the relation between X and Y . Many equations have been proposed to describe the Gilliland curve for multicomponent distillation. Some of these equations are as follows:

- Hengstebeck (1961) correlation²⁵:

$$\log Y = -1.3640187 - 3.0920489Z - 3.407344729Z^2 - 1.74673876Z^3 - 0.33268897Z^4$$

$$Z = \log X$$

- Liddle (1968) correlation²⁶:

$$0.0 \leq X \leq 0.01 : Y = 1.0 - 18.5715X$$

$$0.01 \leq X \leq 0.90: Y = 0.545827 - 0.591422X + \frac{0.002743}{X}$$

$$0.90 \leq X \leq 1.0: Y = 0.16595 - 0.16595X$$

- Van Winkle and Todd (1971) correlation²⁷:

$$0.0078 < X < 0.125: Y = 0.5039 - 0.5968X - 0.0908(\log X)$$

$$0.125 < X \text{ to } X = 1.0: Y = 0.6257 - 0.9868X + 0.516X^2 - 0.1738X^3$$

- Molkavov (1972) correlation²⁸:

$$Y = 1 - \exp\left[\left(\frac{1+54.4X}{11+117.2X}\right)\left(\frac{X-1}{\sqrt{X}}\right)\right] \quad (0 \leq X, Y \leq 1.0)$$

- Hohman and Lockhart (1972) correlation²⁹:

$$X = 0, Y = 0.65; \quad X = 1.0, Y = 0.067$$

$$Y = \frac{0.65 - 0.5X}{1 + 1.25X}$$

- Eduljee (1975) correlation³⁰:

$$X = 0, Y = 1.0; \quad X = 1.0, Y = 0$$

$$Y = 0.75(1 - X^{0.5668})$$

- Chang (1985) correlation³¹:

$$X = 0, Y = 1.0; \quad X = 1.0, Y = 0$$

$$Y = 1 - \exp\left(1.49 + 0.315X - \frac{1.805}{X^{0.1}}\right)$$

- Harg (1985) correlation³²:

$$X = 0, Y = 1.0; \quad X = 1.0, Y = 0$$

$$Y = 1 - X^{1/3}$$

- McCormick (1988) correlation³³:

$$X = 0, Y = 1.0; \quad X = 1.0, Y = 0$$

$$Y = 1 - X^B, \quad B = 0.105(\log X) + 0.44$$

McCormick's correlation gives a good agreement in the normal operating range.

The ratio of the number of stages above the feed stage to the number below the feed stage can be obtained using the Kirkbride equation³⁴:

$$\log\left(\frac{m}{p}\right) = 0.206 \log \left\{ \left(\frac{B}{D} \right) \left(\frac{x_{HK}}{x_{LK}} \right)_F \left[\frac{\left(x_{LK} \right)_B}{\left(x_{HK} \right)_D} \right]^2 \right\}$$

where

m is the number of theoretical stages above the feed stage including the condenser

p is the number of theoretical stages below the feed stage including the reboiler

D and B are the molar flow rates of distillate and bottoms, respectively

7.6.1.4 Feed Stage Location

The location of the optimal feed stage can be estimated by using the Kirkbride equation:

$$\frac{N_R}{N_S} = \left[\left(\frac{Z_{F,HK}}{Z_{F,LK}} \right) \left(\frac{x_{B,LK}}{X_{D,HK}} \right)^2 \left(\frac{B}{D} \right) \right]^{0.206}$$

where

the subscript R denotes the rectifying section

S denotes the stripping section

7.6.1.5 Determination of the Number of Stages by the Smoker Equation

Combination of the vapor–liquid equilibrium relationship

$$y = \frac{\alpha x}{1 + (\alpha - 1)x}$$

and the operating line

$$y = mx + b$$

gives

$$m(\alpha - 1)x^2 + [m + (\alpha - 1)b - \alpha]x + b = 0$$

Let k be the real root between 0 and 1. Then k satisfies the equation

$$m(\alpha - 1)k^2 + [m + (\alpha - 1)b - \alpha]k + b = 0$$

k is the value of the x -ordinate at the point where the extended operating lines intersect the vapor–liquid equilibrium curve. The number of stages required, N , is given by

$$N = \frac{\log \left[\frac{x_0^*(1 - \beta x_n^*)}{x_n^*(1 - \beta x_0^*)} \right]}{\log \left(\frac{\alpha}{mc^2} \right)}$$

where $\beta = mc(\alpha - 1)/(\alpha - mc^2)$.

In the rectifying section, m and b can be expressed in terms of the reflux ratio R :

$$m = \frac{R}{R+1}, \quad b = \frac{x_D}{R+1}$$

m is the slope of the operating line between x_n^* and x_0^* , which are given by

$$x_0 = x_D, \quad x_0^* = x_D - k, \quad x_n = z_F, \quad x_n^* = z_F - k$$

where x_D and z_F are the mole fractions in the distillate and the feed stream, respectively.

In the stripping section, m and b can be represented as

$$m = \frac{Rz_F + x_D - (R+1)x_B}{(R+1)(z_F - x_B)}, \quad b = \frac{(z_F - x_D)x_B}{(R+1)(z_F - x_B)}, \quad x_0^* = z_F - k, \quad x_n^* = x_n - k$$

If the feed stream is not at its bubble point, z_F is replaced by the value of x at the intersection of operating lines given by³⁵

$$z_F^* = \frac{b + \frac{z_F}{q-1}}{\frac{q}{q-1} - m}$$

For distillation at total reflux, the number of stages required is given by

$$N = \frac{\log \left[\frac{x_0(1-x_n)}{x_n(1-x_0)} \right]}{\log \alpha}$$

Example 7.14: Simple Distillation Calculation³⁶

A distillation column is to be used to separate *i*-butane from the mixture of lighter compounds consisting of ethane, propane, *i*-butane, and *n*-butane. The feed stream enters the column as liquid at its bubble point, and the column pressure is 7 bar. It is required that 95% of the *i*-butane in the feed be recovered in the bottoms, and the bottoms stream must contain no more than 0.1% propane.

1. Calculate the minimum number of stages required to achieve the desired separation at total reflux using the Fenske equation.
2. Determine the minimum reflux ratio required to achieve the desired separation with an infinite number of stages using the Underwood equation.
3. Estimate the number of theoretical stages required if the actual reflux ratio is given by $R = 1.5R_m$ using the Gilliland correlation. Assume that the relation between X and Y is represented by the Eduljee correlation. Determine the location of the optimal feed stage using the Kirkbride equation. Table 7.1 shows the feed compositions and the coefficients of the Antoine equation.

Solution

1. The first step of the solution involves the calculation of the distillate composition and bottoms composition. The vapor pressure of each component, P_i , is determined by the

TABLE 7.1
Feed Composition and Antoine Equation Coefficients

Component	Feed Composition (Mole Fraction)	Antoine Equation Coefficients		
		A	B	C
1 Ethane	0.15	3.93835	659.739	-16.719
2 Propane (light key)	0.18	4.53678	1149.36	24.906
3 <i>i</i> -Butane (heavy key)	0.18	4.3281	1132.108	0.918
4 <i>n</i> -Butane	0.49	4.35576	1175.581	-2.071

Antoine equation. The dew point can be obtained using the relationship $\sum x_i = 1 = \sum y_i/k_i$, and the bubble point can be obtained using the relationship $\sum y_i = 1 = \sum k_i x_i$, $k_i = P_i/P$.

2. To calculate the minimum reflux ratio using the Underwood equation, the temperature of the feed stream is determined first and calculate k_{jF} at this temperature. In the evaluation of the equation

$$1 - q = \sum_{j=1}^{n_c} \frac{\alpha_{jF} x_{jF}}{(\alpha_{jF} - \theta)}$$

set $\alpha_{jF} = k_{jF}/k_{HK,F}$ and $q = 1$ because the feed is saturated liquid.

3. We use

$$R = 1.5R_{\min}, \quad X = \frac{R - R_{\min}}{R + 1}, \quad Y = \frac{N - N_{\min}}{N + 1}, \quad Y = 0.75(1 - X^{0.5668})$$

The value of $h = m/p$ can be obtained using the Kirkbride equation. Then we have $N = m + p = (1 + h)p$, $p = N/(1 + h)$, $m = ph$

The script *fugdist* performs the calculation procedure described above.

```
% fugdist.m
% 1: ethane, 2: propane(LK), 3: i-butane(HK), 4: n-butane
xF = [0.15 0.18 0.18 0.49]; % feed composition (mole fraction)
A = [3.93835 4.53678 4.3281 4.35576];
B = [659.739 1149.36 1132.108 1175.581];
C = [-16.719 24.906 0.918 -2.071];
P = 7;
% Compositions of D and B (D: distillate, B: bottom)
Br(1) = 0; Br(3) = 0.95*xF(3); Br(4) = xF(4);
Br(2) = 0.001*(Br(3) + Br(4))/0.999;
Dr(1) = xF(1); Dr(2) = xF(2) - Br(2); Dr(3) = 0.05*xF(3); Dr(4) = 0;
xD = Dr/sum(Dr); xB = Br/sum(Br);
% Dew point(D) and bubble point(B)
Td0 = 270; Tb0 = 300;
fD = @(Td) sum(P*xD./(10.^((A - B./(Td + C)))) - 1;
fB = @(Tb) sum(xB.*10.^((A - B./(Tb + C))/P)) - 1;
Td = fzero(fD, Td0);
Tb = fzero(fB, Tb0);
% Fenske equation
kD = (10.^((A - B./(Td + C))/P));
kB = (10.^((A - B./(Tb + C))/P));
num = log((xD(2)/xD(3))*(xB(3)/xB(2)));
den = log(sqrt((kD(2)/kD(3))*(kB(2)/kB(3))))) ;
Nm = num/den;
% Temperature of feed stream
Tf0 = 300;
```

```

fF = @(Tf) sum(xF.*10.^ (A - B./(Tf + C))/P) - 1;
Tf = fzero(fF, Tf0);
kF = (10.^ (A - B./(Tf + C))/P);
% Underwood equation
th0 = 1.5;
fTh = @(th) sum(xF.*10.^ (A - B./(Tf + C))/P./ (10.^ (A - B./(Tf + C))/P - ...
th*10.^ (A(3) - B(3)/(Tf+C(3)))/P));
theta = fzero(fTh, th0);
alpD = kD/kD(3);
Rm = sum(alpD.*xD./ (alpD - theta)) - 1;
% Gilliland equation
R = 1.5*Rm;
X = (1.5 - 1)*Rm/(R + 1); Y = 0.75*(1 - X^0.5658);
N = (Y+Nm)/(1-Y);
% Kirkbride equation
c1 = sum(Br)/sum(Dr); c2 = xF(3)/xF(2); c3 = (xB(2)/xD(3))^2;
h = (c1*c2*c3)^0.206;
p = N/(1+h); m = p*h;
% Print results
fprintf('Bubble point(B) = %8.4f, Dew point(D) = %8.4f', Tb, Td);
fprintf('\nFeed temp.(F) = %8.4f', Tf);
fprintf('\ntheta = %8.4f, min. number of stages = %7.4f', theta, Nm);
fprintf('\nMin. reflux ratio = %7.4f, actual number of stages = %7.4f', N, Rm);
fprintf('\nStages above the feed stage = %7.4f', m);
fprintf('\nStages below the feed stage = %7.4f\n', p);

```

The script produces the following outputs:

```

>> fugdist
Bubble point(B) = 333.1795, Dew point(D) = 274.0267
Feed temp.(F) = 285.4040
theta = 1.6731, min. number of stages = 8.5438
Min. reflux ratio = 17.3676, actual number of stages = 0.6452
Stages above the feed stage = 3.9784
Stages below the feed stage = 13.3892

```

7.7 RIGOROUS STEADY-STATE DISTILLATION CALCULATIONS

In the rigorous steady-state distillation calculations, the mass balance, energy balance, and phase equilibrium equations for each stage are used. Steady-state distillation calculations consist of solution of the system of nonlinear equations for each stage. [Figure 7.24](#) shows a general equilibrium stage.

In [Figure 7.25](#), V and L are vapor and liquid molar flow rates (lb mol/h), respectively; F is the molar feed rate (lb mol/h); U and W are liquid and vapor sidecuts (lb mol/h), respectively; H and h are vapor and liquid enthalpy (Btu/lb mol), respectively; Q is the heat transfer rate (Btu/h); x and y are the mole fractions of liquid and vapor, respectively; z is the feed composition; the subscript i denotes component; and j denotes stage (numbered from the top to the bottom). Let the number of total stages be n and the number of components be n_c . Balances around the j th stage are as follows:

Component mass balance: $L_{j-1}x_{i,j-1} + V_{j+1}y_{i,j+1} + F_jz_{i,j} = (L_j + U_j)x_{i,j} + (V_j + W_j)y_{i,j}$

Overall mass balance: $L_{j-1} + V_{j+1} + F_j = L_j + U_j + V_j + W_j$

Energy balance: $L_{j-1}h_{j-1} + V_{j+1}H_{j+1} + F_jh_{Fj} = (L_j + U_j)h_j + (V_j + W_j)H_j$

Vapor-liquid equilibrium: $y_{i,j} = k_{i,j}x_{i,j}$

Summation of mole fractions: $\sum_{i=1}^{n_c} y_{i,j} = 1, \sum_{i=1}^{n_c} x_{i,j} = 1$

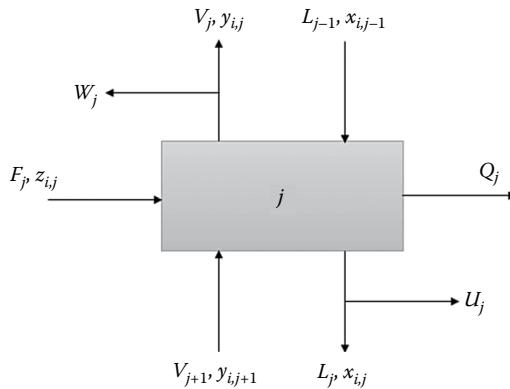


FIGURE 7.25 Equilibrium stage. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 552.)

Example 7.15: Rigorous Distillation Calculations³⁷

A simple distillation column with three theoretical stages shown in Figure 7.26 is to be used to separate *n*-butane(1) and *n*-pentane(2) mixture. A feed stream consisting of 0.23 lb mol/h *n*-butane and 0.77 lb mol/h *n*-pentane enters the column as liquid at its bubble point on stage 2 ($j = 2$). The operating pressure of the column is 120 psia, and a total condenser is used. The amount of heat added to the reboiler is 10,000 Btu/h. Distillate and bottoms are removed at the rates of $D = 0.25$ and $B = 0.75$ lb mol/h, respectively.

Calculate the temperatures of all stages and of the condenser, the flow rates and compositions of vapor and liquid flow in the column, and the compositions of the distillate and the bottoms. The vapor pressures can be estimated by the Antoine equation, and the molar enthalpies of the pure compounds can be estimated by the quadratic equations of T . Table 7.2 shows the Antoine equation coefficients and the enthalpy correlation equations.

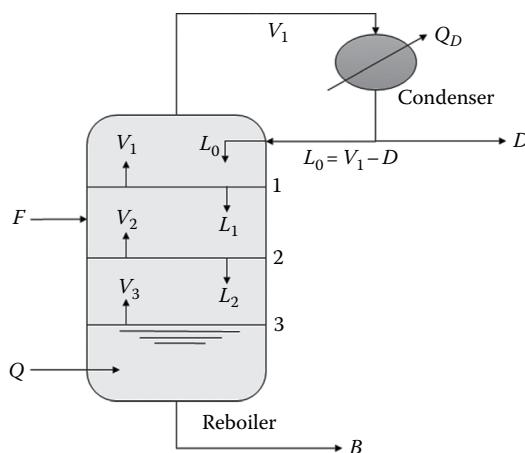


FIGURE 7.26 Three-stage distillation column. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 553.)

TABLE 7.2
Antoine Equation Coefficients and Enthalpy Equations

Component	<i>n</i> -Butane(1)		<i>n</i> -Pentane(2)
Antoine equation coefficients (T : °C)	<i>A</i>	6.80776	6.85296
	<i>B</i>	935.77	1,064.84
	<i>C</i>	238.789	232.012
Enthalpy equation (Btu/lb mol) (T : °F)	Liquid	$h_1 = 0.04T^2 + 29.6T$	$h_2 = 0.025T^2 + 38.5T$
	Vapor	$H_1 = -0.04T^2 + 43.8T + 8003$	$H_2 = 0.007T^2 + 31.7T + 12,004$

Solution

The built-in function *fsolve* can be used to solve the system of nonlinear equations consisting of the component mass balances, the energy balance, and the summations of mole fractions for each stage. There are 14 unknown variables and 14 nonlinear equations:

$$\begin{aligned} \sum_{i=1}^{n_c} k_{i,j} x_{i,j} &= 1 \quad (j = 0, 1, F, 2, 3) \\ -(V_1 - L_0) k_{1,1} + L_1 x_{1,1} + V_2 k_{1,2} x_{1,2} &= 0 \quad (i = 1, 2) \\ L_1 x_{i,1} - (V_2 k_{i,2} + L_2) x_{i,2} + V_3 k_{i,3} x_{i,3} + F z_i &= 0 \quad (i = 1, 2) \\ -V_1 H_1 + V_2 H_2 - L_1 h_1 + L_0 h_0 &= 0 \\ -V_2 H_2 + V_3 H_3 + F h_{ff} + L_1 h_1 - L_2 h_2 &= 0 \\ L_2 x_{i,2} - (V_3 k_{i,3} + B) x_{i,3} &= 0 \quad (i = 1, 2) \\ -V_3 H_3 + Q + L_2 h_2 - L_3 h_3 &= 0 \end{aligned}$$

The function *rdist* defines the nonlinear equations for each stage. The variable vector *s* contains 14 unknown variables as its elements: $s(1) = T_0$, $s(2) = T_1$, $s(3) = T_f$, $s(4) = T_2$, $s(5) = T_3$, $s(6) = x_{11}$, $s(7) = x_{21}$, $s(8) = x_{12}$, $s(9) = x_{22}$, $s(10) = x_{13}$, $s(11) = x_{23}$, $s(12) = V_1$, $s(13) = V_2$, and $s(14) = V_3$. The script *userdist* calls the function *rdist* and employs the function *fsolve* to solve the nonlinear equations.

```
rdist.m
function f = rdist(s,F,D,z,P,Q)
% i=1: n-butane, i=2:n-Pentane)
% s1: T0, s2: T1, s3: Tf, s4: T2, s5: T3, s6: x11, s7: x21
% s8: x12, s9: x22, s10: x13, s11: x23, s12: V1, s13: V2, s14: V3
% Antoine coefficients and parameters
A = [6.80776 6.85296]; B = [935.77 1064.84]; C = [238.789 232.012];
h1c = [0.04 29.6]; h2c = [0.025 38.5];
H1c = [-0.04 43.8 8003]; H2c = [0.007 31.7 12004];
% Equilibrium constants
k0 = 10.^((A - B./((s(1)-32)*5/9 + C)) / P;
k1 = 10.^((A - B./((s(2)-32)*5/9 + C)) / P;
kf = 10.^((A - B./((s(3)-32)*5/9 + C)) / P;
k2 = 10.^((A - B./((s(4)-32)*5/9 + C)) / P;
k3 = 10.^((A - B./((s(5)-32)*5/9 + C)) / P;
% Composition vector
x1 = [s(6) s(7)]; x2 = [s(8) s(9)]; x3 = [s(10) s(11)];
x0 = k1.*x1;
% Enthalpy
hL0 = sum(([s(1)^2 s(1)]*[h1c' h2c']).*x0);
```

```

hL1 = sum(([s(2)^2 s(2)]*[h1c' h2c']).*x1);
hLf = sum(([s(3)^2 s(3)]*[h1c' h2c']).*z);
hL2 = sum(([s(4)^2 s(4)]*[h1c' h2c']).*x2);
hL3 = sum(([s(5)^2 s(5)]*[h1c' h2c']).*x3);
hV1 = sum(([s(2)^2 s(2) 1]*[H1c' H2c']).*k1.*x1);
hV2 = sum(([s(4)^2 s(4) 1]*[H1c' H2c']).*k2.*x2);
hV3 = sum(([s(5)^2 s(5) 1]*[H1c' H2c']).*k3.*x3);
% Mass balance
B = F - D;
L0 = s(12) - D;
L1 = s(13) - D;
L2 = s(14) + B;
L3 = B;
% Definition of equations
f(1) = sum(k0.*x0) - 1;
f(1, 2) = sum(k1.*x1) - 1;
f(1, 3) = sum(kf.*z) - 1;
f(1, 4) = sum(k2.*x2) - 1;
f(1, 5) = sum(k3.*x3) - 1;
f(1, 6) = -(s(12) - L0)*k1(1) + L1)*s(6) + s(13)*k2(1)*s(8);
f(1, 7) = -(s(12) - L0)*k1(2) + L1)*s(7) + s(13)*k2(2)*s(9);
f(1, 8) = -s(12)*hV1 + s(13)*hV2 - L1*hL1 + L0*hL0;
f(1, 9) = L1*s(6) - (s(13)*k2(1)+L2)*s(8) + s(14)*k3(1)*s(10) + F*z(1);
f(1, 10) = L1*s(7) - (s(13)*k2(2)+L2)*s(9) + s(14)*k3(2)*s(11) + F*z(2);
f(1, 11) = -s(13)*hV2 + s(14)*hV3 + hLf + L1*hL1 - L2*hL2;
f(1, 12) = L2*s(8) - (s(14)*k3(1) + B)*s(10);
f(1, 13) = L2*s(9) - (s(14)*k3(2) + B)*s(11);
f(1, 14) = -s(14)*hV3 + Q + L2*hL2 - L3*hL3;
end

userdist.m
% userdist.m
% s1: T0, s2: T1, s3: Tf, s4: T2, s5: T3, s6: x11, s7: x21
% s8: x12, s9: x22, s10: x13, s11: x23, s12: V1, s13: V2, s14: V3
F = 1; D = 0.25; z = [0.23 0.77]; P = 760*120/14.7; Q = 1e4;
T0 = 200; T1 = 145; Tf = 200; T2 = 190; T3 = 210; x11 = 0.65; x21 = 0.35;
x12 = 0.43; x22 = 0.57; x13 = 0.33; x23 = 0.76; V1 = 1.1; V2 = 1; V3 = 1.1;
s0 = [T0 T1 T2 T3 x11 x21 x12 x22 x13 x23 V1 V2 V3];
s = fsolve(@rdist,s0,[],F,D,z,P,Q);
T0 = s(1); T1 = s(2); Tf = s(3); T2 = s(4); T3 = s(5);
x11 = s(6); x21 = s(7); x12 = s(8); x22 = s(9); x13 = s(10); x23 = s(11);
V1 = s(12); V2 = s(13); V3 = s(14);
fprintf('T0=%8.4f, T1=%8.4f, Tf=%8.4f, T2=%8.4f, T3=%8.4f\n',T0,T1,Tf,T2,T3);
fprintf('x11=%6.4f, x12=%6.4f, x13=%6.4f\n', x11, x12, x13);
fprintf('x21=%6.4f, x22=%6.4f, x23=%6.4f\n', x21, x22, x23);
fprintf('V1=%6.4f, V2=%6.4f, V3=%6.4f\n', V1, V2, V3);

```

The script userdist produces the following outputs:

```

>> userdist
T0=188.2848, T1=206.4285, Tf=215.4895, T2=218.5461, T3=226.5515
x11=0.3272, x12=0.1991, x13=0.1225
x21=0.6728, x22=0.8009, x23=0.8775
V1=1.0537, V2=1.0388, V3=1.0417

```

The results can be summarized as shown in Table 7.3.

7.7.1 RIGOROUS DISTILLATION MODEL: MESH EQUATIONS

For a steady-state stage, we assume that phase equilibrium is achieved, no chemical reactions occur, and entrainment of liquid drops in vapor and occlusion of vapor bubbles in liquid are negligible. Figure 7.27 represents a general equilibrium stage. There are vapor and liquid sidestreams, and heat can be transferred at a rate of Q_j from (+) or to (-) the stage j .

TABLE 7.3
Calculation Results for a Three-Stage Column

Variable	Stage 1	Feed Stage	Stage 2	Stage 3
Temperature (°F)	206.4285	215.4895	218.5461	226.5515
Vapor flow rate (lb mol/h)	1.0537	—	1.0388	1.0417
Composition of <i>n</i> -butane	0.3272	0.23	0.1991	0.1225
Composition of <i>n</i> -pentane	0.6728	0.77	0.8009	0.8775

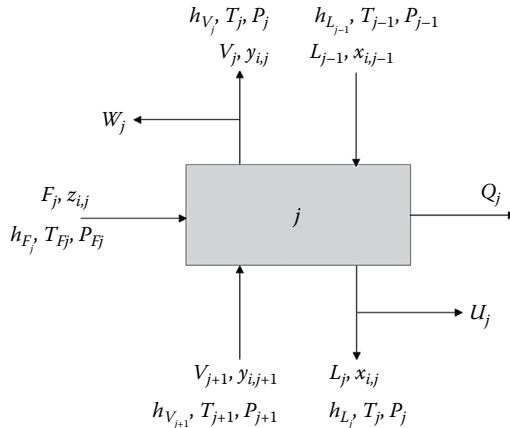


FIGURE 7.27 A steady-state equilibrium stage model. (From Henley, E.J. et al., *Separation Process Principles*, 3rd ed., John Wiley & Sons, Hoboken, NJ, 2011, p. 411.)

Entering stage j is a feed stream of molar flow rate F_j with composition in mole fraction $z_{i,j}$ and molar enthalpy h_F . Feed pressure is equal to stage pressure P_j . Interstage liquid from stage $j - 1$ of molar flow rate L_{j-1} is also entering to stage j as well as interstage vapor from stage $j + 1$ of molar flow rate V_{j+1} . Leaving stage j is vapor of properties $y_{i,j}$, h_V , T_j , and P_j . The vapor steam may be divided into a vapor sidestream of molar flow rate W_j and an interstage stream of molar flow rate V_j to be sent to stage $j - 1$. Also leaving stage j is liquid of properties $x_{i,j}$, h_L , T_j , and P_j in equilibrium with vapor. This liquid may be divided into a sidestream of molar flow rate U_j and an interstage stream of molar flow rate L_j to be sent to stage $j + 1$.

Figure 7.28 shows a distillation column with N equilibrium stages that are numbered down from the top. There are N stages and the number of components in the feed mixture is C . Stage 1 ($j = 1$) is the partial condenser and stage N ($j = N$) is the reboiler.

Let the amount of component i in the j th stage be $X_{i,j}$. Using the relationship $y_{i,j} = v_{i,j}/V_j$ and $x_{i,j} = l_{i,j}/L_j$, the mass balances for the j th stage yield

$$M_{i,j} = V_{j+1}y_{i,j+1} + L_{j-1}x_{i,j-1} + F_jz_{i,j} - (V_j + W_j)y_{i,j} - (L_j + U_j)x_{i,j} = 0$$

$$M_{i,j} = v_{i,j+1} + l_{i,j-1} + f_{i,j} - (1 + S_{V_j})v_{i,j} - (1 + S_{L_j})l_{i,j} = 0$$

where $S_{V_j} = W_j/V_j$ and $S_{L_j} = U_j/L_j$.

The phase equilibrium equation for each component gives

$$E_{i,j} = y_{i,j} - K_{i,j}x_{i,j} = \frac{v_{i,j}}{V_j} - K_{i,j} \frac{l_{i,j}}{L_j} = 0$$

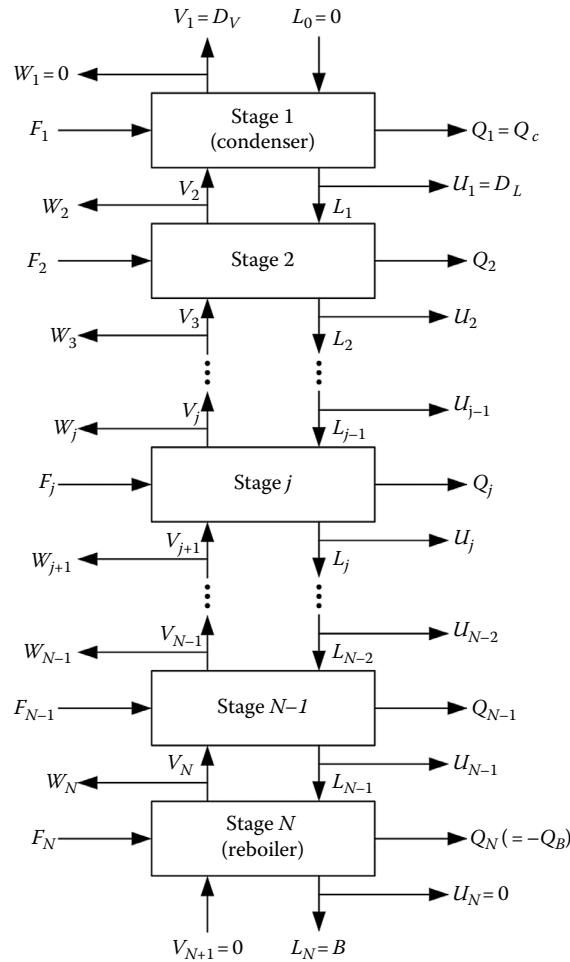


FIGURE 7.28 A distillation column with N stages. (From Henley, E.J. et al., *Separation Process Principles*, 3rd ed., John Wiley & Sons, Hoboken, NJ, 2011, p. 411.)

The summation of mole fractions for each stage should be equal to 1:

$$S_{y,j} = \sum_{i=1}^C y_{i,j} - 1 = \sum_{i=1}^C \frac{v_{i,j}}{V_j} - 1 = 0, \quad S_{x,j} = \sum_{i=1}^C x_{i,j} - 1 = \sum_{i=1}^C \frac{l_{i,j}}{L_j} - 1 = 0$$

The energy balance for each stage gives

$$H_j = L_{j-1}h_{L_{j-1}} + V_{j+1}h_{V_{j+1}} + F_jh_{F_j} - (L_j + U_j)h_{L_j} - (V_j + W_j)h_{V_j} - Q_j = 0$$

Using the summation of component flow relations $V_j = \sum_{i=1}^C v_{i,j}$ and $L_j = \sum_{i=1}^C l_{i,j}$ and $F_j = \sum_{i=1}^C f_{i,j}$, this equation becomes

$$H_j = h_{V_{j+1}} \sum_{i=1}^C v_{i,j+1} + h_{L_{j-1}} \sum_{i=1}^C l_{i,j-1} + h_{F_j} \sum_{i=1}^C f_{i,j} - (1 + S_{V_j})h_{V_j} \sum_{i=1}^C v_{i,j} - (1 + S_{L_j})h_{L_j} \sum_{i=1}^C l_{i,j} - Q_j = 0$$

For each equilibrium stage, there are 2C+3 MESH (Mass, Equilibrium, Summations, and Enthalpy) equations.

The total mass balance can be obtained from the summation of component mass balance equations from stage 1 to stage j :

$$\sum_{k=1}^j \sum_{i=1}^C M_{i,k} = L_0 - L_j + V_{j+1} - V_1 + \sum_{k=1}^j (F_k - U_k - W_k) = 0$$

Since $L_0 = 0$, this equation can be rewritten as

$$L_j = V_{j+1} - V_1 + \sum_{k=1}^j (F_k - U_k - W_k) \quad (1 \leq j \leq N-1)$$

The summation of component mass balances from stage j to stage N gives

$$\sum_{k=j}^N \sum_{i=1}^C M_{i,k} = V_{N+1} - V_j + L_{j-1} - L_N + \sum_{k=j}^N (F_k - U_k - W_k) = 0$$

Since $V_{N+1} = 0$, this equation can be expressed as

$$V_j = L_{j-1} - L_N + \sum_{k=j}^N (F_k - U_k - W_k)$$

7.7.2 TRIDIAGONAL MATRIX METHOD

The tridiagonal matrix method introduced by Wang and Henke³⁸ is a fast and accurate technique for calculating the component and total flow rates. In the tridiagonal matrix method, the calculation for a stage is based on the calculation for the previous stage, and the accumulation of calculation errors can be significant when the number of stages is large. In order to avoid the accumulation of calculation errors, the revised Thomas algorithm can be used. In the Thomas algorithm, the calculation begins from the upper left corner of the matrix (i.e., the first stage). Similar calculations are repeated along the diagonal direction of the matrix.

Calculation of the stage liquid composition $x_{i,j}$: The mass balances and equilibrium equations give

$$\begin{aligned} & \left\{ V_j - V_1 + \sum_{k=1}^{j-1} (F_k - U_k - W_k) \right\} x_{i,j-1} \\ & - \left\{ (V_j + W_j) K_{i,j} + V_{j+1} - V_1 + \sum_{k=1}^j (F_k - U_k - W_k) + U_j \right\} x_{i,j} + V_{j+1} K_{i,j+1} x_{i,j+1} = -F_j z_{i,j} \end{aligned}$$

This equation can be rewritten as

$$P_j x_{i,j-1} + Q_j x_{i,j} + R_j x_{i,j+1} = S_j$$

where

$$P_j = V_j - V_1 + \sum_{k=1}^{j-1} (F_k - U_k - W_k) \quad (2 \leq j \leq N)$$

$$Q_j = V_1 - (V_j + W_j)K_{i,j} - U_j - V_{j+1} - \sum_{k=1}^j (F_k - U_k - W_k) \quad (1 \leq j \leq N)$$

$$R_j = V_{j+1}K_{i,j+1} \quad (1 \leq j \leq N-1)$$

$$S_j = -F_j z_{i,j} \quad (1 \leq j \leq N)$$

$$x_{i,N+1} = x_{i,0} = 0, \quad V_{N+1} = W_1 = U_N = 0$$

Rearrangement of the equations for all stages ($j = 1, 2, \dots, N$) yields

$$\begin{bmatrix} 1 & r_1 & 0 & \cdots & 0 \\ 0 & 1 & r_2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & 1 & r_{N-1} \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,N-1} \\ x_{i,N} \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{N-1} \\ s_N \end{bmatrix}$$

where

$$r_j = \frac{R_j}{Q_j - r_{j-1}P_j} \quad (2 \leq j \leq N-1), \quad r_1 = \frac{R_1}{Q_1}$$

$$s_j = \frac{S_j - s_{j-1}P_j}{Q_j - r_{j-1}P_j} \quad (2 \leq j \leq N), \quad s_1 = \frac{S_1}{Q_1}$$

The liquid composition $x_{i,j}$ for stage j is given by

$$r_0 = s_0 = 0, \quad x_{i,N} = s_N, \quad x_{i,j} = s_j - r_j x_{i,j+1} \quad (1 \leq j \leq N-1)$$

Calculation of the stage vapor component flow rate $v_{i,j}$: The calculation procedure for $v_{i,j}$ is similar to the previous one. The absorption factor $A_{i,j}$ can be defined as $A_{i,j} = L_j/K_{i,j}V_j$.

Then, since $l_{i,j} = L_j/K_{i,j}V_j = A_{i,j}v_{i,j}$, the component mass balance gives

$$M_{i,j} = v_{i,j+1} + A_{i,j-1}v_{i,j-1} + f_{i,j} - (1 + S_{V_j})v_{i,j} - (1 + S_{L_j})A_{i,j}v_{i,j} = 0$$

This equation can be rewritten as

$$A_{i,j-1}v_{i,j-1} + B_jv_{i,j} + v_{i,j+1} = D_j$$

where

$$B_j = -1 - S_{V_j} - (1 + S_{L_j})A_{i,j}, \quad D_j = -f_{i,j}$$

Collecting equations for all stages yields

$$\begin{bmatrix} 1 & p_1 & 0 & \cdots & 0 \\ 0 & 1 & p_2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & 1 & p_{N-1} \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{i,1} \\ v_{i,2} \\ \vdots \\ v_{i,N-1} \\ v_{i,N} \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_{N-1} \\ q_N \end{bmatrix}$$

where

$$p_j = \frac{1}{B_i - p_{i-1}A_{i,j-1}} \quad (1 \leq j \leq N-1), \quad q_j = \frac{D_j - q_{j-1}A_{i,j-1}}{B_j - p_{j-1}A_{i,j-1}} \quad (1 \leq j \leq N)$$

and $A_{i,0} = p_0 = q_0 = 0$. From these equations, the component vapor flow rate is given by

$$v_{i,N} = q_N, \quad v_{i,j} = q_j - p_j v_{i,j+1} \quad (1 \leq j \leq N-1)$$

Calculation of the stage liquid component flow rate $l_{i,j}$: The calculation procedure for $l_{i,j}$ is similar to the previous one for $v_{i,j}$. The relative volatility and the stripping factor $S_{i,j}$ can be expressed as

$$\alpha_{i,j} = \frac{K_{i,j}}{K_{b,j}}, \quad S_{b,j} = \frac{K_{b,j}V_j}{L_j}, \quad S_{i,j} = \frac{K_{i,j}V_j}{L_j} = \alpha_{i,j}S_{b,j}$$

where the subscript b denotes the reference component. If we define R_L and R_V as

$$R_{L_j} = 1 + \frac{U_j}{L_j} = 1 + S_{L_j}, \quad R_{V_j} = 1 + \frac{W_j}{V_j} = 1 + S_{V_j}$$

the component vapor flow rate can be represented as

$$v_{i,j} = \frac{K_{i,j}V_j}{L_j} l_{i,j} = \frac{K_{i,j}}{K_{b,j}} \frac{K_{b,j}V_j}{L_j} l_{i,j} = \alpha_{i,j} S_{b,j} l_{i,j}$$

Combination of these equations yields

$$l_{i,j-1} + W_j l_{i,j} + Z_j l_{i,j+1} = G$$

where

$$W_j = -R_{L_j} - \alpha_{i,j} S_{b,j} R_{V_j}, \quad Z_j = \alpha_{i,j+1} S_{b,j+1}, \quad G_j = -f_{i,j}$$

Collecting the balance equations for all stages gives

$$\begin{bmatrix} 1 & p_1 & 0 & \cdots & 0 \\ 0 & 1 & p_2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & 1 & p_{N-1} \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} l_{i,1} \\ l_{i,2} \\ \vdots \\ l_{i,N-1} \\ l_{i,N} \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_{N-1} \\ q_N \end{bmatrix}$$

where

$$p_j = \frac{Z_j}{W_j - p_{j-1}} \quad (1 \leq j \leq N-1), \quad q_j = \frac{G_j - q_{j-1}}{W_j - p_{j-1}} \quad (1 \leq j \leq N), \quad p_0 = q_0 = 0$$

Consequently, the component liquid flow rates are given by

$$l_{i,N} = q_N, \quad l_{i,j} = q_j - p_j l_{i,j+1} \quad (1 \leq j \leq N-1)$$

7.7.3 BUBBLE-POINT (BP) METHOD

In the BP method, a form of the equilibrium and summation equations is used to calculate the stage temperature. A new set of stage temperatures is calculated during each iteration from the bubble-point equations. The BP method generally works best for narrow-boiling, ideal, or nearly ideal systems, where composition has a greater effect on temperature than the latent heat of vaporization.³⁹

The condenser load Q_1 and the reboiler load Q_N can be expressed as

$$Q_1 = (L_1 + V_1 + U_1 + W_1 - F_1)h_{V2} - V_1h_{V1} - (L_1 + U_1)h_{L1} + F_1h_{F1}$$

$$Q_N = \sum_{j=1}^N (F_jh_{Fj} - U_jh_{Lj} - W_jh_{Vj}) - \sum_{i=1}^{N-1} Q_j - V_1h_{V1} - L_Nh_{LN}$$

The liquid flow rate L_j and the vapor flow rate V_j at the j th stage are given by

$$L_j = V_{j+1} - V_1 + \sum_{k=1}^j (F_k - U_k - W_k) \quad (1 \leq j \leq N-1), \quad L_0 = 0$$

$$V_{j+1} = \frac{d_j - a_j V_j}{b_j} \quad (2 \leq N \leq j-1), \quad V_2 = L_1 + V_1 + U_1 + W_1 - F_1$$

$$a_j = h_{L_{j-1}} - h_{V_j}, \quad b_j = h_{V_{j+1}} - h_{L_j}$$

$$d_j = (h_{L_j} - h_{L_{j-1}}) \sum_{k=1}^{j-1} (F_k - U_k - W_k) + F_j (h_{L_j} - h_{Fj}) + W_j (h_{Vj} - h_{Lj}) + Q_j$$

In the calculation procedure of the BP method, the liquid composition $x_{i,j}$ and the vapor composition $y_{i,j}$ are normalized as

$$x_{i,j} = \frac{x_{i,j}}{\sum_i^C x_{i,j}}, \quad y_{i,j} = \frac{y_{i,j}}{\sum_{i=1}^C y_{i,j}}$$

The solution is considered converged when sets of stage temperatures at the k th iteration are within some prescribed tolerance of corresponding sets of stage temperatures at the $k-1$ th iteration. A simple criterion based on successive sets of stage temperatures can be used for the convergence test:

$$J_T = \sum_{j=1}^N \left\{ T_j^{(k)} - T_j^{(k-1)} \right\}^2 \leq \epsilon N \quad (0 < \epsilon < 1)$$

The BP algorithm can be summarized as follows:

1. Initialize the input variables (F_j , $z_{i,j}$, T_F , P_F , h_F , P_j , U_j , W_j , Q_j [excluding Q_1 and Q_N], N , L_1 [reflux flow rate], V_1). Set $k = 1$.
2. Set the initial values of the tear variables T_j and V_j .
3. Calculate $x_{i,j}$ using $x_{i,N} = S_N$, $x_{i,j} = s_j - r_j x_{i,j+1}$ ($1 \leq j \leq N-1$), $r_0 = s_0 = 0$, and normalize the resulting $x_{i,j}$ by $x_{i,j} = x_{i,j} / \sum_{i=1}^C x_{i,j}$. This procedure has to be done for all components.

4. Use the bubble-point calculation method to find a new set of T_j and $y_{i,j}$. The equilibrium equation $E_{i,j} = y_{i,j} - K_{i,j}x_{i,j} = 0$ is used in the calculation.
5. Calculate the condenser load Q_1 and the reboiler load Q_N :

$$Q_1 = (L_1 + V_1 + U_1 + W_1 - F_1)h_{V2} - V_1h_{V1} - (L_1 + U_1)h_{L1} + F_1h_{F1}$$

$$Q_N = \sum_{j=1}^N (F_jh_{Fj} - U_jh_{Lj} - W_jh_{Vj}) - \sum_{j=1}^{N-1} Q_j - V_1h_{V1} - L_Nh_{LN}$$

6. Calculate V_j and L_j sequentially:

$$V_{j+1} = \frac{d_j - a_j V_j}{b_j} \quad (2 \leq N \leq j-1), \quad V_2 = L_1 + V_1 + U_1 + W_1 - F_1$$

$$L_j = V_{j+1} - V_1 + \sum_{k=1}^j (F_k - U_k - W_k) \quad (1 \leq j \leq N-1), \quad L_0 = 0$$

7. Check the convergence using the criterion $J_T = \sum_{j=1}^N \left\{ T_j^{(k)} - T_j^{(k-1)} \right\}^2 \leq \epsilon N$. If the set of stage temperatures does not converge, adjust the tear variables again, set $k = k + 1$, and go to step 3.

The MATLAB function *distBPeu* performs this calculation procedure (in the function name, eu denotes use of the English unit system). The basic syntax is

```
[x,y,T,L,V,iter] = distBPeu(opdat,mxdat)
```

The structure *opdat* contains operating conditions as fields, and the structure *mxdat* contains physical properties and parameters as its fields. This function calls other function *phiHeu* to calculate the enthalpy of the mixture and the fugacity of each component.

```
function [x,y,T,L,V,iter] = distBPeu(opdat,mxdat)
% Calculation of multicomponent distillation using BP method (English unit)
% Set data
eos = opdat.eos; N = opdat.N; nc = opdat.nc;
F = opdat.F; Tf = opdat.Tf; Pf = opdat.Pf; hF = opdat.hF;
P = opdat.P; T = opdat.T; V = opdat.V; L = opdat.L; criv = opdat.criv;
U = opdat.U; W = opdat.W; Q = opdat.Q; T0 = opdat.T0; nf = opdat.nf;
hV = opdat.hV; hL = opdat.hL; fstate = upper(opdat.fstate);
x = opdat.x; y = opdat.y; z = opdat.z;
Pc = mxdat.Pc; Ant = mxdat.Ant;
% Initialization
a = zeros(1,N); b = zeros(1,N); d = zeros(1,N);
V(2) = L(1) + V(1) + U(1) + W(1) - F(1);
% Feed enthalpy
for j = 1:length(nf)
    [Z H phi] = phiHeu(z(:,nf(j))',Pf(nf(j)),Tf(nf(j)),fstate,eos,opdat,
    mxdat);
    hF(nf(j)) = H; % Btu/mol
end
```

```

T = T0; Told = T; crit = 10; iter = 1;
% Initialization of K(i,j): K(i,j) at T0
for j = 1:N
    Pv0(:,j) = Pc(:).*exp(Ant(:,1) - Ant(:,2)./(T(j) + Ant(:,3)));
    K(:,j) = Pv0(:,j)./P(j); % Raoult's law
end
while crit > critv
    % Calculate x(i,j)
    for i = 1:nc
        for j = 2:N-1
            Pj(j) = V(j) - V(1) + sum(F(1:j-1) - U(1:j-1) - W(1:j-1));
            Qj(j) = V(1) - (V(j) + W(j))*K(i,j) - U(j) - V(j+1) - ...
            sum(F(1:j) - U(1:j) - W(1:j));
            Rj(j) = V(j+1)*K(i,j+1); Sj(j) = -F(j)*z(i,j);
        end
        Pj(N) = V(N) - V(1) + sum(F(1:N-1) - U(1:N-1) - W(1:N-1));
        Qj(1) = V(1) - (V(1) + W(1))*K(i,1) - U(1) - V(2) - (F(1) - U(1) ...
        - W(1));
        Qj(N) = V(1) - (V(N) + W(N))*K(i,N) - U(N) - sum(F(1:N) - U(1:N) ...
        - W(1:N));
        Rj(1) = V(2)*K(i,2); Sj(1) = -F(1)*z(i,1); Sj(N) = -F(N)*z(i,N);
        r(1) = Rj(1)/Qj(1); s(1) = Sj(1)/Qj(1);
        for j = 2:N-1
            r(j) = Rj(j)/(Qj(j) - r(j-1)*Pj(j));
            s(j) = (Sj(j) - s(j-1)*Pj(j))/(Qj(j) - r(j-1)*Pj(j));
        end
        s(N) = (Sj(N) - s(N-1)*Pj(N))/(Qj(N) - r(N-1)*Pj(N));
        x(i,N) = s(N);
        for j = N-1:-1:1
            x(i,j) = s(j) - r(j)*x(i,j+1);
        end
    end
    for j = 1:N
        x(:,j) = x(:,j)/sum(x(:,j));
    end
    % Calculate new T(j) and y(i,j) using BP method
    for j = 1:N % calculate y(i,j)
        y(:,j) = K(:,j).*x(:,j);
        y(:,j) = y(:,j)/sum(y(:,j));
    end
    % Antoine eqn. (T: deg.F)
    for j = 1:N % calculate T(j) (T in F)
        f = @(Tv) sum(Pc'.*exp(Ant(:,1) - Ant(:,2)./(Tv + Ant(:,3))).*x(:,j) / ...
        P(j)) - 1;
        T(j) = fzero(f,T(j));
    end
    for j = 1:N
        lstate = 'L';
        [Z H phi] = phiHeu(x(:,j)',P(j),T(j),lstate eos,opdat,mxdat);
        phiL = phi; hL(j) = H; % Btu/lbmol
        vstate = 'V';
        [Z H phi] = phiHeu(y(:,j)',P(j),T(j),vstate eos,opdat,mxdat);
        phiV = phi; hV(j) = H; % Btu/lbmol
        K(:,j) = phiL./phiV;
    end
    % Calculate loads for the condenser and reboiler

```

```

Q(1) = (L(1) + V(1) + U(1) + W(1) - F(1))*hV(2) - V(1)*hV(1) - ...
       (L(1) + U(1))*hL(1) + F(1)*hF(1);
Q(N) = sum(F.*hF - U.*hL - W.*hV) - sum(Q(1:N-1)) - ...
       V(1)*hV(1) - L(N)*hL(N);
% Calculate V(j) and L(j) sequentially
for j = 2:N
    a(j) = hL(j-1) - hV(j);
    b(j-1) = hV(j) - hL(j-1);
    d(j) = (hL(j)-hL(j-1))*sum(F(1:j-1)-U(1:j-1)-W(1:j-1)) + ...
            F(j)*(hL(j)-hF(j)) + W(j)*(hV(j)-hL(j)) + Q(j);
end
a(1) = -hV(1); b(N) = -hL(N);
d(1) = F(1)*(hL(1)-hF(1)) + W(1)*(hV(1)-hL(1)) + Q(1);
for j = 2:N-1
    V(j+1) = (d(j) - a(j)*V(j))/b(j);
    L(j) = V(j+1) - V(1) + sum(F(1:j) - U(1:j) - W(1:j));
end
% Check convergence
crit = sum(abs(Told - T));
Told = T; iter = iter + 1;
end
end

```

The function *phiHeu* calculates the enthalpy of the mixture and the fugacity coefficient of each component using the English unit system. The basic syntax is

```
[Z H phi] = phiHeu(x,P,T,state,eos,opdat,mxdat)
```

where *x* is the mole fraction vector, *P* and *T* are pressure (psia) and temperature (°F), respectively, *state* denotes the fluid state ('L': liquid, 'V': vapor), and *eos* is the equation of state being used ('RK', 'SRK', or 'PR'). *opdat* is a structure containing operating conditions as its fields. The structure *mxdat* includes physical properties and parameters as its fields: *mxp.Pc* is a critical pressure (psia) vector, *mxp.Tc* is a critical temperature (R) vector, *mxp.w* is a vector of acentric factors, *mxp.k* is a matrix of binary interaction parameters (n x n diagonal matrix), and *mxp.Afi* is a matrix of heat capacity integration coefficients (dimension: number of components x number of parameters). In the output arguments, *Z* is the compressibility factor, *H* is the mixture enthalpy (Btu/lb mol), and *phi* is a vector denoting the fugacity coefficients of each component.

```

function [Z H phi] = phiHeu(x,P,T,state,eos,opdat,mxdat)
% Calculation of mixture enthalpy and fugacity coefficient of component
% Inputs:
% x: mole fraction vector
% P, T: pressure (Psia) and temperature (F)
% state: fluid state ('L': liquid, 'V': vapor)
% eos: equation of state ('RK', 'SRK', or 'PR')
% mxp: physical property structure
% mxp.Pc, mxp.Tc: critical pressure (Psia) and temperature (R) vector
% mxp.w: acentric factor for each component (vector)
% mxp.k: nxn diagonal matrix of binary interaction parameters
% mxp.Afi: heat capacity integration coefficients for ideal gases
% Outputs:
% Z: compressibility factor
% H: mixture enthalpy (Btu/lbmol)
% phi: fugacity coefficient
w = mxdat.w; k = mxdat.k;
Tc = mxdat.Tc/1.8; Pc = 6894.8*mxdat.Pc; % T and P in K and Pa

```

```

T = (T-32)/1.8 + 273.15; P = 6894.8*P;
Afi = mxdat.Afi; % Btu/lbmole(1Btu/lbmole = 2.326 J/mol)
nc = opdat.nc;
R = 8.314; % gas constant: m^3 Pa/(mol K) = J/mol-K
Tr = T./Tc; Pr = P./Pc; nc = opdat.nc;
eos = upper(eos); state = upper(state);
switch eos
case{'RK'}
ai = sqrt(0.4278./(Pc.*Tr.^2.5)); bi = 0.0867./(Pc.*Tr);
A = sum(x.*ai); B = sum(x.*bi);
Z = roots([1 -1 B*B*(A^2/B-B*B-1) -A^2*(B*B)^2/B]);
case{'SRK'}
mx = 0.48+1.574*w-0.176*w.^2; al = (1+mx.*(1-sqrt(Tr))).^2;
ai = 0.42747*al.*Pr./(Tr.^2); bi = 0.08664*Pr./Tr;
am = sqrt(ai'*ai).* (1-k); A = x*am*x'; B = sum(x.*bi);
Z = roots([1 -1 A-B-B^2 -A*B]);
case{'PR'}
mx = 0.37464+1.54226*w-0.26992*w.^2; al = (1+mx.*(1-sqrt(Tr))).^2;
ai = 0.45723553*al.*Pr./(Tr.^2); bi = 0.0777961*Pr./Tr;
am = sqrt(ai'*ai).* (1-k); A = x*am*x'; B = sum(x.*bi);
Z = roots([1 B-1 A-3*B^2-2*B B^3+B^2-A*B]);
end
iz = abs(imag(Z)); Z(and(iz>0,iz<=1e-6)) = real(Z(and(iz>0,iz<=1e-6)));
for i = 1:length(Z), zind(i) = isreal(Z(i)); end
Z = Z(zind);
if state == 'L'
    Z = min(Z);
else
    Z = max(Z);
end
V = R*T*Z/P; % m^3/mol
% Enthalpy and fugacity coefficients
% Hv0 = int(Cp) Btu/lbmol, Afi uses T in F
Te = (T-273.15)*1.8 + 32; % T: K->F
Hv0 = x*(Afi(:,1)*Te + Afi(:,2)*Te^2/2 + Afi(:,3)*Te^3/3 + ...
Afi(:,4)*Te^4/4 + Afi(:,5)*Te^5/5);
Hv0 = Hv0*2.326; % 1Btu/lbmole=2.326J/mol
switch eos
case{'RK'}
    H = Hv0 + R*T*(Z - 1 - 3*(A^2)*log(1 + B*B/Z)/(2*B));
    phi = exp((Z-1)*bi/B - log(Z-B*B) - (A^2/B)*(2*ai/A - bi/B)*log(1 + B*B/Z));
case{'SRK'}
    hsum = 0;
    for i = 1:nc
        for j = 1:nc
            hsum = hsum + x(i)*x(j)*am(i,j)*(1 - ...
            mx(i)*sqrt(Tr(i))/(2*sqrt(al(i))) - ...
            mx(j)*sqrt(Tr(j))/(2*sqrt(al(j))));
        end
    end
    H = Hv0 + R*T*(Z - 1 - log((Z + B)/Z)*hsum/B);
    phi = exp((Z-1)*bi/B - log(Z-B) - (A/B)*(2*sqrt(ai)/sqrt(A) - bi/B)*log((Z+B)/Z));
case{'PR'}
    hsum = 0;

```

```

for i = 1:nc
    for j = 1:nc
        hsum = hsum + x(i)*x(j)*am(i,j)*(1 -
            mx(i)*sqrt(Tr(i))/(2*sqrt(al(i))) - ...
            mx(j)*sqrt(Tr(j))/(2*sqrt(al(j)))) ;
    end
end
H = Hv0 + R*T*(Z - 1 - log((Z + B)/Z)*hsum/B) ;
phi = exp((Z-1)*bi/B - log(Z-B) - (A/B/sqrt(8))*(2*x(1:end)*am(1:
end,:)/A - ...
bi/B)*log((Z+(1+sqrt(2))*B)/(Z+(1-sqrt(2))*B))) )
end
H = H/2.326; % J/mol -> Btu/lbmol
end

```

Example 7.16: Distillation of a Five-Component Mixture

Figure 7.29 shows a distillation column where a feed consisting of ethane(1)/propane(2)/n-butane(3)/n-pentane(4)/n-hexane(5) is distilled. The number of total stages including the condenser and the reboiler is 16. The feed stream 1 enters to the first feed stage (6th stage from the top) at 170°F and 300 psia with the flow rate of $F_6 = 41$ lb mol/hr and the composition of $z_{1,6} = 0.061$, $z_{2,6} = 0.342$, $z_{3,6} = 0.463$, $z_{4,6} = 0.122$, and $z_{5,6} = 0.012$. The feed stream 2 enters to the second feed stage (9th stage from the top) at 230°F and 275 psia with the flow rate of $F_9 = 59$ lb mol/hr and the composition of $z_{1,9} = 0.0085$, $z_{2,9} = 0.1017$, $z_{3,9} = 0.3051$, $z_{4,9} = 0.5085$, and $z_{5,9} = 0.0762$. The pressure in the column is kept constant as 240 psia and the flow rate of the vapor stream to the first stage (V_1)

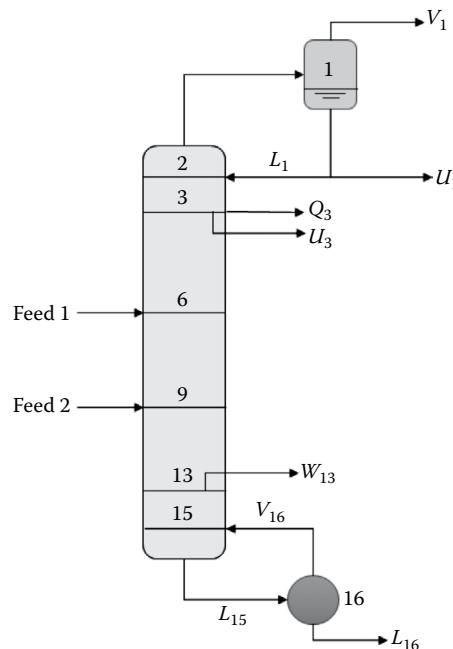


FIGURE 7.29 A simple 16-stage distillation column. (From Henley, E.J. et al., *Separation Process Principles*, 3rd ed., John Wiley & Sons, Hoboken, NJ, 2011, p. 411.)

TABLE 7.4
Physical Properties of Feed Mixture

Component	P_c (psia)	T_c (R)	u	A_1	A_2	A_3
Ethane $C_2H_6(1)$	709.8	550.0	0.1064	5.38389	2847.921	434.898
Propane $C_3H_8(2)$	617.4	665.9	0.1538	5.35342	3371.084	414.488
<i>n</i> -Butane $C_4H_{10}(3)$	550.7	765.3	0.1954	5.74162	4126.385	409.5179
<i>n</i> -Pentane $C_5H_{12}(4)$	489.5	845.9	0.2387	5.85365	4598.287	394.4148
<i>n</i> -Hexane $C_6H_{14}(5)$	440.0	914.2	0.2972	6.03924	5085.758	382.794

is 15 lb mol/hr and that of the liquid stream to the first stage (L_1) is 150 lb mol/hr. The top product is obtained as liquid from the first stage and the flow rate of the liquid sidecut is $U_1 = 5$ lb mol/hr. An intercooler is implemented in the 3rd stage and the cooling rate is $Q_3 = 200,000$ Btu/hr. The flow rate of liquid sidecut from the 3rd stage is $U_3 = 3$ lb mol/hr and that of vapor sidecut from the 13th stage is $W_{13} = 37$ lb mol/hr.

The specific heat C_{pv}^0 (Btu/lb mol) and the vapor pressure (P_i^S) are given by

$$C_{pv}^0 = a_1 + a_2 T + a_3 T^2 + a_4 T^3 + a_5 T^4 \quad (T: {}^\circ F)$$

$$\ln \frac{P_i^S}{P_c} = A_1 - \frac{A_2}{T + A_3} \quad (T: {}^\circ F, P_c : \text{critical pressure})$$

Table 7.4 shows the physical properties and Table 7.5⁴⁰ shows parameters of specific heat equations for each component. Calculate the stage temperatures and vapor and liquid compositions and plot the results versus the stage number.⁴¹

Solution

The script *c5distBP.m* specifies the parameters for each component and operating conditions and calls the function *distBP* to perform the BP calculations. The *K*-value for the component *i* is obtained from $K_i = \phi_{il}/\phi_{iv}$ and the enthalpy and the fugacity coefficients for vapor and liquid phases in each stage are calculated by the function *phiHeu*.

```
% c5distBP.m: calculation of 5-component distillation using BP method
% (English unit)
% Operating conditions and parameters
```

TABLE 7.5
Parameters of the Specific Heat Equation for Each Component

Component	a_1	$a_2 \times 10$	$a_3 \times 10^4$	$a_4 \times 10^7$	$a_5 \times 10^{11}$
Ethane $C_2H_6(1)$	11.51606	0.140309	0.085404	-0.110608	0.316220
Propane $C_3H_8(2)$	15.58683	0.250495	0.140426	-0.352626	1.864467
<i>n</i> -Butane $C_4H_{10}(3)$	20.79783	0.314329	0.192851	-0.458865	2.380972
<i>n</i> -Pentane $C_5H_{12}(4)$	25.64627	0.389176	0.239729	-0.584262	3.079918
<i>n</i> -Hexane $C_6H_{14}(5)$	30.17847	0.519926	0.030488	-0.27640	1.346731

```

clear all;
opdat.N = 16; opdat.nc = 5; % N: total stages, nc: number of components
intv = zeros(1,opdat.N); intc = zeros(opdat.nc,opdat.N);
opdat.F = intv; opdat.Tf = intv; opdat.Pf = intv; opdat.hF = intv;
opdat.P = intv; opdat.T = intv; opdat.V = intv; opdat.L = intv;
opdat.U = intv; opdat.W = intv; opdat.Q = intv; opdat.eos = 'pr';
opdat.hV = intv; opdat.hL = intv; opdat.nf = 0;
opdat.x = intc; opdat.y = intc; opdat.z = intc;
% Physical properties and parameters for each component
mxdat.Pc = [709.8 617.4 550.7 489.5 440.0]; % critical pressure (psia)
mxdat.Tc = [550.0 665.9 765.3 845.9 914.2]; % critical temperature (R)
mxdat.k = zeros(opdat.nc,opdat.nc); % matrix of binary interaction parameters
mxdat.w = [0.1064 0.1538 0.1954 0.2387 0.2972]; % acentric factors
% Antoine equation parameters
mxdat.Ant = [5.38389 2847.921 434.898; 5.35342 3371.084 414.488;
5.74162 4126.385 409.5179; 5.853654 4598.287 394.4148;
6.03924 5085.758 382.794];
% Specific heat parameters (a(i))
mxdat.Afi = [11.51606 0.140309e-1 0.0854034e-4 -0.110608e-7 0.316220e-11;
15.58683 0.2504953e-1 0.1404258e-4 -0.352626e-7 1.864467e-11;
20.79783 0.314329e-1 0.192851e-4 -0.458865e-7 2.380972e-11;
25.64627 0.389176e-1 0.239729e-4 -0.584262e-7 3.079918e-11;
30.17847 0.519926e-1 0.030488e-4 -0.27640e-7 1.346731e-11];
% Operating conditions (P: psia, T: F, flow: lbmol/hr)
opdat.nf = [6 9]; % feed stage
opdat.fstate = 'V'; % feed state
opdat.F(opdat.nf) = [41 59]; opdat.V(1) = 15; opdat.L(1) = 150;
opdat.U(1) = 5; opdat.U(3) = 3; opdat.W(13) = 37;
opdat.L(opdat.N) = sum(opdat.F(opdat.nf)) - opdat.V(1) - opdat.U(1) - ...
opdat.U(3) - opdat.W(13);
opdat.Pf(opdat.nf) = [300 275]; opdat.Tf(opdat.nf) = [170 230];
opdat.P = 240*ones(1,opdat.N); opdat.Q(3) = 2e5;
opdat.z(:,opdat.nf(1)) = [0.061 0.342 0.463 0.122 0.012];
opdat.z(:,opdat.nf(2)) = [0.0085 0.1017 0.3051 0.5085 0.0762];
% Initialization and guess
opdat.T0 = (300/opdat.N)*[1:opdat.N]; % guess stage temperatures (F)
opdat.V(2:opdat.N-1) = 170;
opdat.criv = 1e-3; % convergence criterion
% BP calculation
[x,y,T,L,V,iter] = distBPeu(opdat,mxdat);
fprintf('Number of iterations (convergence criterion: %g): %d\n', opdat.criv, iter)
disp('Temperature: ')
Nx = 1:opdat.N;
x1 = x(1,:); x2 = x(2,:); x3 = x(3,:); x4 = x(4,:); x5 = x(5,:);
y1 = y(1,:); y2 = y(2,:); y3 = y(3,:); y4 = y(4,:); y5 = y(5,:);
figure(1), plot(Nx,T), xlabel('Stage'), ylabel('T(F)'), axis tight
figure(2), plot(Nx,x1,Nx,x2,'--',Nx,x3,'.-',Nx,x4,:',Nx,x5,'*')
xlabel('Stage'), ylabel('x'), legend('x_1','x_2','x_3','x_4','x_5')
figure(3), plot(Nx,y1,Nx,y2,'--',Nx,y3,'.-',Nx,y4,:',Nx,y5,'*')
xlabel('Stage'), ylabel('y'), legend('y_1','y_2','y_3','y_4','y_5')

```

The script *c5distBPeu* calculates the number of iterations and temperatures, and generates the temperature and composition profiles as shown in [Figures 7.30 through 7.32](#):

```

>> c5distBPeu
Number of iterations (convergence criterion: 0.001): 40
Temperature:
T =
102.0861 112.6251 118.4383 124.3199 132.5619 146.0313 157.7010 173.5853
195.1363 205.4129 215.8198 226.3022 237.5966 249.2418 262.1954 276.7688

```

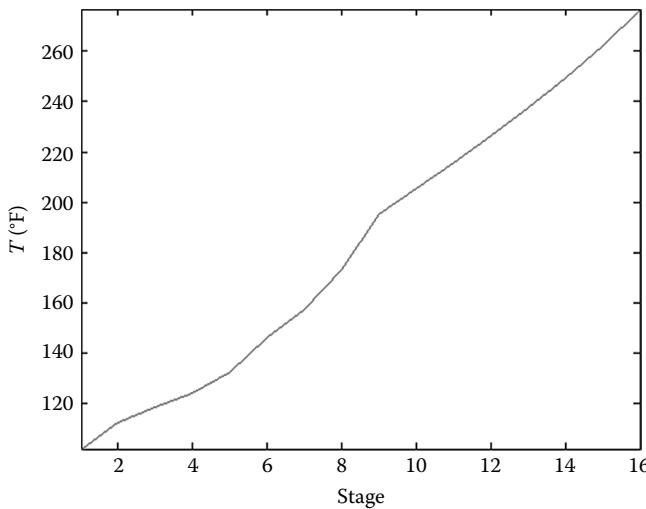


FIGURE 7.30 Temperature profile for the 16-stage column.

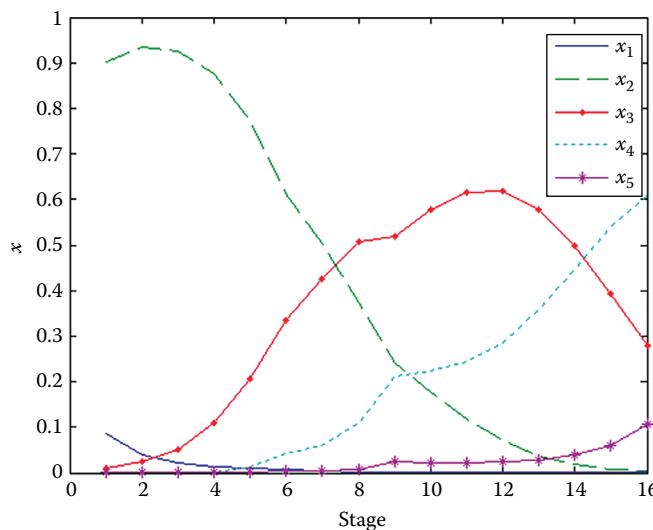


FIGURE 7.31 Liquid composition profile for the 16-stage column.

7.8 DIFFERENTIAL DISTILLATION

In simple differential distillation, the liquid is heated slowly to the boiling point and the vapor is collected and condensed. [Figure 7.33](#) shows a simple single-stage batch distillation apparatus. At the start, the vapor condensed contains the richest fraction of the more volatile component. But, as the vaporization continues, its concentration decreases.

The concentration of the more volatile component, x , in the liquid solution (L) changes through time. Thus, we can formulate the equation using the mass balance based on the more volatile component. Let dx be the change of the concentration of the more volatile component in the liquid

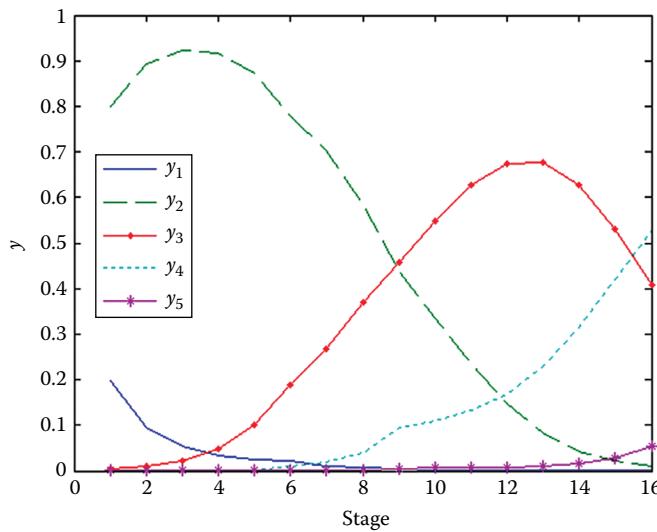


FIGURE 7.32 Vapor composition profile for the 16-stage column.

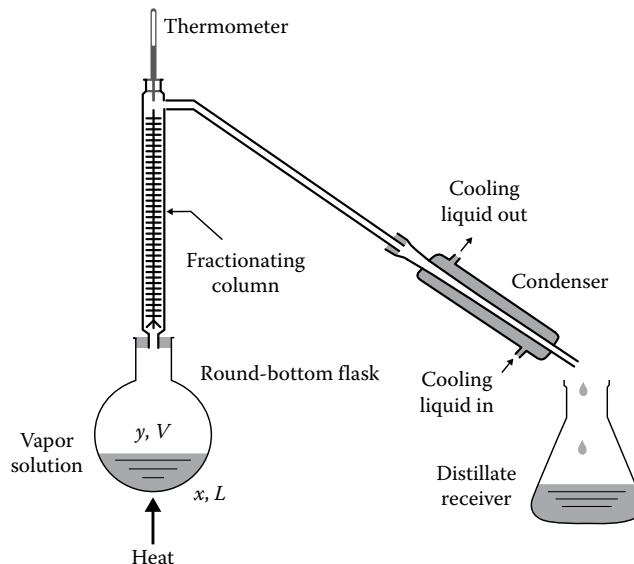


FIGURE 7.33 A simple single-stage batch distillation apparatus.

solution, dL be the fractional amount of the liquid evaporated, and y be the concentration of the more volatile component in the vapor (V). The balance equation gives

$$xL = (x - dL)(L - dL) + y dL = xL - L dx - x dL + dx dL + y dL$$

The magnitude of the term $dx dL$ is negligible compared to other terms in this equation. Then we have

$$xL = xL - L dx - x dL + y dL$$

Rearrangement of this equation yields

$$-\frac{dL}{L} = \frac{dx}{x-y} \quad \text{or} \quad \frac{dL}{dx} = \frac{L}{y-x}$$

Integration of both sides of this equation gives

$$-\int_{L_1}^{L_2} \frac{dL}{L} = \int_{x_1}^{x_2} \frac{dx}{x-y} \Rightarrow -\ln \frac{L_2}{L_1} = \int_{x_1}^{x_2} \frac{dx}{x-y}$$

where

L_1 is the initial number of moles of the liquid solution

L_2 is the number of moles of liquid after evaporation takes place

Example 7.17: Differential Distillation

A solution containing 70 mol of benzene and 50 mol of toluene is distilled using the simple differential distillation apparatus at 1 atm until only 50 mol of liquid is left. Calculate the composition of the liquid after the distillation process. [Table 7.6](#) shows the phase equilibrium data for benzene–toluene mixture.

Solution

$$x_1 = \frac{70}{70+50} = 0.5833, \quad -\ln \frac{L_2}{L_1} = -\ln \frac{50}{120} = 0.8755$$

Thus, the value of x_2 satisfying

$$-\ln \frac{L_2}{L_1} = 0.8755 = \int_{0.5833}^{x_2} \frac{dx}{x-y} = \int_{0.5833}^{x_2} \frac{dx}{f(x)}$$

is to be determined. The built-in function `quad` can be used for the integration. The given equilibrium data can be regressed using 4th-order polynomial.

```
>> x = [0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0];
>> y = [0.000 0.211 0.378 0.512 0.623 0.714 0.791 0.856 0.911 0.959 1.000];
>> p = polyfit(x,y,4);
>> f = @(x) 1./((x-(p(1)*x.^4+p(2)*x.^3+p(3)*x.^2+p(4)*x+p(5)));
>> x1 = 0.5833; vfix = 0.8755; crit = 1e-2;
>> for x2 = x1:-1e-3:0.0, if abs(quad(f,x1,x2)-vfix) <= crit, break, end
end
>> xf = x2

xf =
0.3983
```

We can see that the benzene composition in the residue solution after the distillation operation is $x_f = 0.3983$.

TABLE 7.6

Phase Equilibrium Data for Benzene–Toluene Mixture

x	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y	0.000	0.211	0.378	0.512	0.623	0.714	0.791	0.856	0.911	0.959	1.000

Example 7.18: Single-Stage Batch Distillation⁴²

A single-stage batch distillation apparatus is used for separation of ethanol (2) from water (1). A liquid mixture of 40 mol% water ($x_1 = 0.4$) and 60 mol% ethanol ($x_2 = 0.6$) is charged initially to the still pot. The amount of the initial charge is 100 kg mol ($L_1 = 100$). The distillation is carried out at 1 atm total pressure and is continued until the water mole fraction reaches 0.8 ($x_1 = 0.8$). Plot the temperature profile and the amount of the liquid remaining in the still as a function of the water composition. Calculate the temperature and the composition of the liquid remaining in the still after the distillation process. The vapor-liquid equilibrium follows the Raoult's law: $k_i = \gamma_i P_i / P$. The vapor pressure P_i is given by the Antoine equation and the activity coefficient of component i , γ_i , is given by

$$\log \gamma_1 = (1 - x_1)^2 \{c_1 + 2x_1(c_2 - c_1)\}, \quad \log \gamma_2 = (1 - x_2)^2 \{c_2 + 2x_2(c_1 - c_2)\}$$

where $c_1 = 0.3781$ and $c_2 = 0.6848$. The temperature change with respect to the liquid composition may be represented by

$$\frac{dT}{dx_1} = K(1 - k_1 x_1 - K_2 x_2) \quad (K : \text{constant})$$

The constant K may take a large value such as 500,000. The Antoine equation parameters for water and ethanol are given in Table 7.7.

Solution

The function *btdist* defines the system of algebraic differential equations. The script *usebtdist* calls the function *btdist* and employs the built-in function *ode45* to find solutions.

```
btdist.m
function dz = btdist(x1,z,Pt,A,B,C,c,K)
% z(1) = T, z(2) = L
x2 = 1-x1; x = [x1 x2];
Pj = 10.^((A - B./(z(1) + C)));
gam(1) = 10.^((1-x1)^2*(c(1) + 2*x1*(c(2)-c(1))));
gam(2) = 10.^((1-x2)^2*(c(2) + 2*x2*(c(1)-c(2))));
k = gam.*Pj/Pt;
dz(1) = K*(1 - k(1)*x1 - k(2)*x2);
dz(1, 2) = z(2)/(x1*(k(1)-1));
end

usebtdist.m
% usebtdist.m
% % z(1) = T, z(2) = L
A = [7.96681 8.04494]; B = [1668.21 1554.3]; C = [228 222.65];
c = [0.3781 0.6848]; Pt = 760; K = 5e5; x1v = [0.4 0.8]; z0 = [79 100];
```

TABLE 7.7
Antoine Equation Parameters for Water and Ethanol
 $(T: {}^\circ\text{C}, P: \text{mmHg})$

Component	Composition (Mole Fraction)	Antoine Equation Parameters		
		A	B	C
1 Water	0.4	7.96681	1668.21	228
2 Ethanol	0.6	8.04494	1554.3	222.65

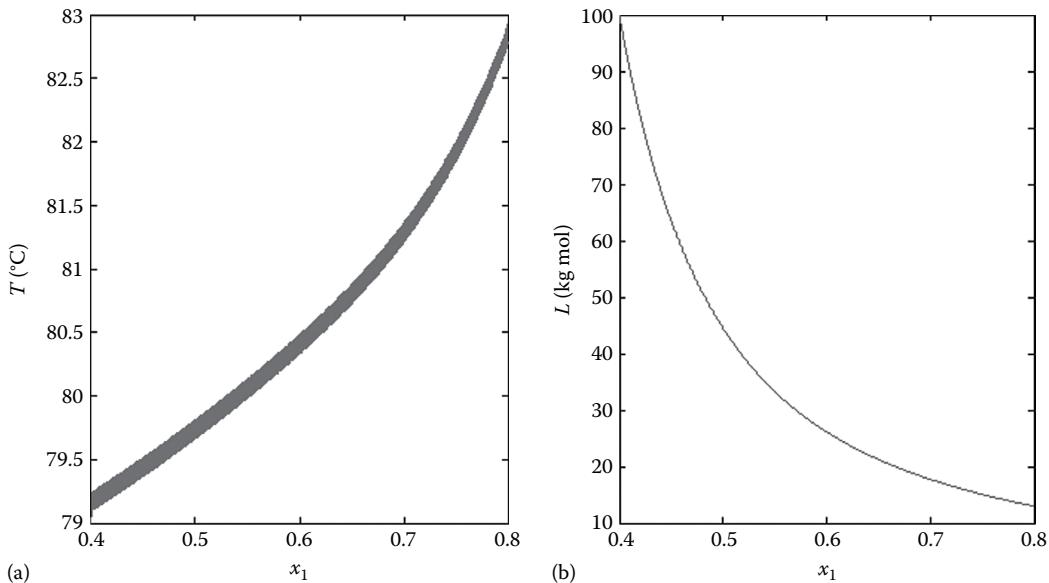


FIGURE 7.34 (a) Temperature and (b) liquid profiles in the batch distillation process.

```
[x1 z] = ode45(@btdist,x1v,z0,[],Pt,A,B,C,c,K);
T = z(:,1); L = z(:,2); fprintf('Tfinal = %g, Lfinal = %g\n',T(end),L(end))
subplot(1, 2), plot(x1,T), xlabel('x_1'), ylabel('T(C)')
subplot(1, 2), plot(x1,L), xlabel('x_1'), ylabel('L(kg mole)')
```

The script usebtdist generates the following results and the plots shown in [Figure 7.34](#):

```
>> usebtdist
Tfinal = 82.7562, Lfinal = 13.0123
```

We can see that the final temperature is 82.76°C and the amount of the liquid remaining in the still is 13.01 kg mol .

7.9 MEMBRANE SEPARATION

In membrane processes for gas separation, a gas phase is present on both sides of the membrane, which is usually a polymer such as rubber, polyamide, and so on. The solute gas first dissolves in the membrane and then diffuses in the solid to the other gas phase. High-pressure feed gas is supplied to one side of the membrane and permeates normal to the membrane. The permeate leaves in a direction normal to the membrane, accumulating on the low-pressure side. Because of the very high diffusion coefficient in gases, concentration gradients in the gas phase in the direction normal to the surface of the membrane are very small. Hence, gas film resistances compared to the membrane resistance may be neglected.

There are several cases in the operation of a membrane module. Separate theoretical models can be derived for different types of operation. In deriving models for gas separation by membranes, isothermal conditions and negligible pressure drop in the feed stream are assumed. It is also assumed that pressure drop in the gas stream can be calculated by using the Hagen–Poiseuille equation and that the permeability of each component is constant (i.e., no interactions between different components).

7.9.1 COMPLETE-MIXING MODEL FOR GAS SEPARATION

In the complete-mixing model, changes in compositions of gas streams are assumed to be negligible. Figure 7.35 shows a flow diagram for complete-mixing model.

The overall material balance is given by

$$L_f = L_R + V_p$$

where

L_f is the molar flow rate of feed stream

L_R is the molar flow rate of reject stream

V_p is the molar flow rate of permeate stream

The cut or the fraction of the feed permeated, θ , is expressed as

$$\theta = \frac{V_p}{L_f}$$

For a binary feed consisting of component A and component B , the rate of permeation of component A is given by

$$\frac{q_A}{A_m} = \frac{V_p y_p}{A_m} = \left(\frac{P_A}{t} \right) (p_h x_R - p_l y_p)$$

where

P_A is the permeability of A in the membrane

q_A is the flow rate of A in permeate

A_m is the membrane area

p_h is the total pressure in the high-pressure side

p_l is the total pressure in the low-pressure side

x_R is the mole fraction of A in reject

y_p is the mole fraction of A in permeate

t is the membrane thickness

The rate of permeation of component B is given by

$$\frac{q_B}{A_m} = \frac{V_p (1 - y_p)}{A_m} = \left(\frac{P_B}{t} \right) [p_h (1 - x_R) - p_l (1 - y_p)]$$

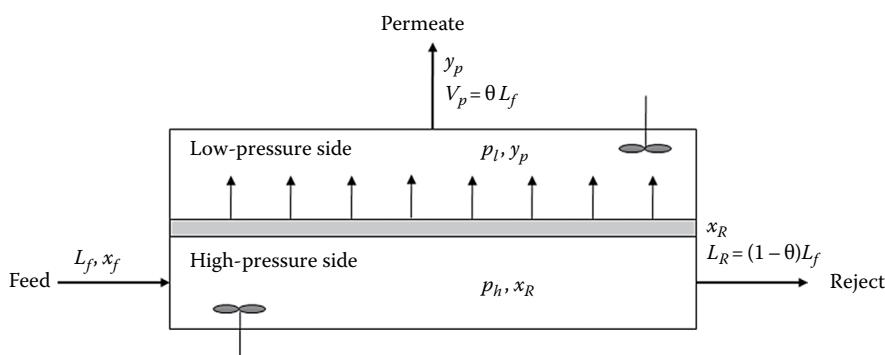


FIGURE 7.35 Complete-mixing flow pattern.

where P_B is the permeability of B in the membrane. From these two equations, we have

$$y_p \{ (1 - x_R) - r(1 - y_p) \} = \alpha(1 - y_p)(x_R - ry_p)$$

where $\alpha = P_A/P_B = P_{CO_2}/P_{N_2}$ and $r = p/p_h$. Solving this equation for y_p yields

$$y_p = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

where $a = 1 - \alpha$, $b = -1 + \alpha + (1/r) + (\alpha - 1)(x_R/r)$, and $c = (-\alpha x_R/r)$. The concentrations of the reject and permeate streams as well as the cut are given by

$$x_R = \frac{x_f - \theta y_p}{(1 - \theta)}, \quad y_p = \frac{x_f - (1 - \theta)x_R}{\theta}, \quad \theta = \frac{x_f - x_R}{y_p - x_R}$$

The membrane area can be expressed as

$$A_m = \frac{\theta L_f y_p}{(P_A/t)(p_h x_R - p_l y_p)}$$

The recovery, r_c , defined by the ratio of the amount of A permeated to the amount of A supplied, is given by

$$r_c = \frac{V_p y_p}{L_f x_f} \times 100$$

If all the feed supplied is permeated (i.e., $V_p = L_f$), $\theta = 1$ and $y_p = x_f$. Hence,

$$x_R = \frac{x_f \{ 1 + r(\alpha - 1)(1 - x_f) \}}{x_f (1 - \alpha) + \alpha}$$

This equation represents the minimum reject composition. In calculations, two cases can be considered: (1) x_f , x_R , α , r are given and y_p , θ , A_m are to be determined; (2) x_f , θ , α , r are given and y_p , x_R , A_m are to be determined.

The function *cpmlex* calculates the membrane separation process based on the complete-mixing model for binary feed. The syntax is

```
res = cpmlex
```

This function calls the script *cpmdata* that specifies membrane data and operating conditions.

```
function res = cpmlex
cpmdata;
Pa = Pm(1); a = 1 - alpa; b = -1 + alpa + 1./r + (alpa-1)*xr./r;
c = -alpa*xr./r; yp = (-b+sqrt(b.^2 - 4*a.*c))./(2*a); % permeate mole fraction
theta = (xf-xr)./(yp-xr); % stage-cut
Am = (theta.*qf.*yp)./((Pa./t).* (ph.*xr - pl.*yp)); % membrane area
rc = qf*theta*yp/(qf*xf); % recovery ratio
xom = (xf*(1+r*(alpa-1)*(1-xf)))/(xf*(1-alpa)+alpa);
% Results: res = [yp, xr, Am, theta, rc]
res.yp = yp; res.theta = theta; res.Am = Am; res.rc = rc; res.xr = xr;
```

Example 7.19: Complete-Mixing Model⁴³

A membrane is to be used to separate a gaseous mixture of *A* and *B*. The feed flow rate is $L_f = 1 \times 10^4 \text{ cm}^3/\text{s}$ and the feed composition of *A* is $x_f = 0.5$ (mole fraction). The desired composition of the reject is $x_R = 0.25$. Calculate the permeate composition y_p , the stage-cut θ , and the membrane area A_m .

Data: membrane thickness $t = 2.54 \times 10^{-3} \text{ cm}$, feed pressure $p_h = 80 \text{ cmHg}$, permeate-side pressure $p_l = 20 \text{ cmHg}$

Permeability of *A*: $P_A = 50 \times 10^{-10} \text{ cm}^3 \cdot \text{cm}/(\text{s} \cdot \text{cm}^2 \cdot \text{cmHg})$

Permeability of *B*: $P_B = 5 \times 10^{-10} \text{ cm}^3 \cdot \text{cm}/(\text{s} \cdot \text{cm}^2 \cdot \text{cmHg})$

Solution

The script *cpmdata* specifies the membrane data and operating conditions.

```
% cpmdata.m: gas mixture and membrane data
t = 0.00254; % membrane thickness (cm)
Pm = [50 5]*1e-10; % permeability (cm^3*cm/(s*cm^2*cmHg))
alpa = Pm(1)/Pm(2);
ph = 80; % feed-side pressure (cmHg)
pl = 20; % permeate-side pressure (cmHg)
r = pl/ph; % pressure ratio (Plow/Phigh)
qf = 1e4; % feed flow rate (cm^3/s(STP))
xf = 0.5; % feed composition (mole fraction)
xr = 0.25; % desired reject composition (mole fraction)
```

The function *cpm1ex* produces the following results:

```
>> cpm1ex
ans =
yp: 0.6036
theta: 0.7071
Am: 2.7344e+08
rc: 0.8535
xr: 0.2500
```

7.9.2 CROSS-FLOW MODEL FOR GAS SEPARATION

In the cross-flow pattern, the velocity of the high-pressure gas stream is large enough that this stream is in plug flow and flows parallel to the membrane. The flow of the permeate stream in the low-pressure side is essentially perpendicular to the membrane. It is assumed that there is no mixing in both the low-pressure and high-pressure sides. Thus, the permeate composition at any point along the membrane is determined by the relative rates of permeation of the feed components at that point. Figure 7.36 shows a flow diagram for cross-flow model.

Rearranging the equations representing the local permeation rate over a differential membrane area dA_m at any point gives

$$\frac{y}{1-y} = \frac{\alpha(x-ry)}{(1-x)-r(1-y)}$$

where $\alpha = P_A/P_B = P_{\text{CO}_2}/P_{\text{N}_2}$, $r = p_l/p_h$. The analytical solution was given by Weller and Steiner as⁴⁴

$$\frac{(1-\theta^*)(1-x)}{(1-x_f)} = \left(\frac{u_f - E/D}{u - E/D} \right)^R \left(\frac{u_f - \alpha + F}{u - \alpha + F} \right)^S \left(\frac{u_f - F}{u - F} \right)^T$$

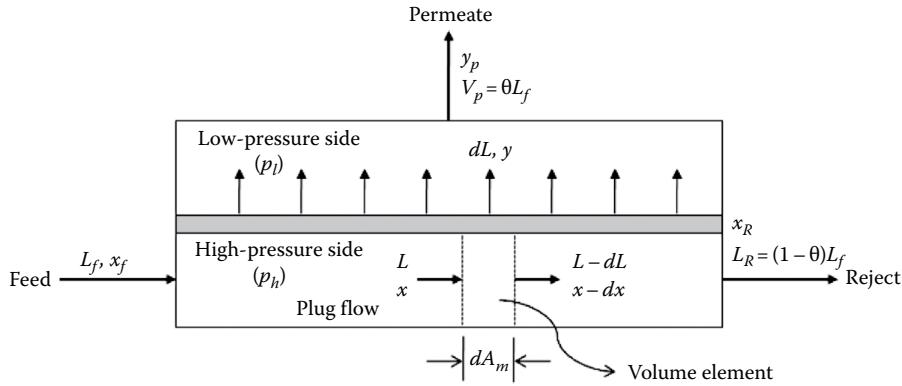


FIGURE 7.36 Cross-flow pattern.

where

$$\theta^* = \frac{L_f - L}{L_f} = 1 - \frac{L}{L_f}, \quad i = \frac{x}{1-x} \left(\text{or } x = \frac{i}{i+1} \right), \quad u = -Di + \sqrt{D^2 i^2 + 2Ei + F^2}$$

$$D = \frac{1}{2} [(1-\alpha)r + \alpha], \quad E = \frac{\alpha}{2} - DF, \quad F = -\frac{1}{2} [(1-\alpha)r - 1], \quad R = \frac{1}{2D-1}$$

$$S = \frac{\alpha(D-1) + F}{(2D-1)(\alpha/2 - F)}, \quad T = \frac{1}{1-D-(E/F)}$$

u_f denotes the value of u at $i = i_f = x_f/(1-x_f)$ and θ^* represents the fraction permeated up to x . At the outlet, $x = x_R$ and θ^* is equal to the total permeation fraction θ . The total membrane area, A_m , may be obtained from the equation

$$A_m = \frac{L_f}{p_h (P_B/t)} \int_{i_R}^{i_f} \frac{(1-x_i) \left(\frac{u_f - E/D}{u - E/D} \right)^R \left(\frac{u_f - \alpha + F}{u - \alpha + F} \right)^S \left(\frac{u_f - F}{u - F} \right)^T}{(f_i - i) \left(\frac{1}{1+i} - \frac{r}{1+f_i} \right)} di$$

where $i_f = x_f/(1-x_f)$, $i_R = x_R/(1-x_R)$.

The function *crflex* calculates the membrane separation process based on the cross-flow model for a binary feed mixture. The basic syntax is

```
res = crflex
```

This function calls the script *crfdata* that specifies membrane data and operating conditions. The function *mArea* defines the function to be integrated to give the membrane area.

crflex.m

```
function res = crflex
% Membrane separation process for binary feed using cross-flow model
clear all;
crfdata;
Pa = Pm(1); Pb = Pm(2); D1 = ((1-alpa)*r + alpa)/2;
F1 = -((1-alpa)*r - 1)/2; E1 = alpa/2 - D1.*F1;
R1 = 1./(2*D1-1); S1 = (alpa.* (D1-1) + F1)./( (2*D1 - 1).* (alpa/2 - F1));
```

```

T1 = 1./(1 - D1 - E1./F1);
i0 = xr./(1-xr); i2 = xf./(1-xf);
ur = -D1.*i0 + sqrt((D1.^2).*i0.^2 + 2*E1.*i0 + F1.^2);
uf = -D1.*i2 + sqrt((D1.^2).*i2.^2 + 2*E1.*i2 + F1.^2);
theta = 1 - ((1-xf)./(1-xr)).*((uf-E1./D1)./(ur-E1./D1)).^R1...
.*((uf-alpa+F1)./(ur-alpa+F1)).^S1 .*((uf-F1)./(ur-F1)).^T1;
yp = (xf - (1-theta)*xr)/theta;
Ami = quad(@(x) mArea(x,D1,E1,F1,R1,S1,T1,alpa,r,xf,uf), i0, i2);
Am = Ami*qf*t/(ph*Pb); rc = theta*yp./xf; % recovery ratio
% Calculated variables: yp(permeate mole fraction), Am(area)
% theta(stage-cut) or xr(reject composition)
% Results: results = [yp, xr, Am, theta, rc]
res.yp = yp; res.theta = theta; res.Am = Am; res.rc = rc; res.xr = xr;
end

```

mArea.m

```

function y = mArea(x,D,E,F,R,S,T,alpa,r,xf,uf)
u = -D.*x + sqrt((D.^2).* (x.^2) + 2*E.*x + F.^2);
fi = (D.*x - F) + sqrt((D.^2)*(x.^2) + 2*E.*x + F.^2);
ud = (1-xf).*((uf-E/D)./(u-E/D)).^R.*((uf-alpa+F)./(u-alpa+F)).^S...
.*((uf-F)./(u-F)).^T;
y = ud./((fi-x).* (1./(1+x) - r./(1+fi)));
end

```

Example 7.20: Cross-Flow Model⁴⁵

A membrane is to be used to separate a gaseous mixture of *A* and *B*. The feed flow rate is $L_f = 1 \times 10^4 \text{ cm}^3/\text{sec}$ and the feed composition of *A* is $x_f = 0.5$ (mole fraction). The desired composition of the reject is $x_R = 0.25$. Use the cross-flow model to determine the permeate composition y_p , the stage-cut θ , and the membrane area A_m .

Data: membrane thickness $t = 2.54 \times 10^{-3} \text{ cm}$, feed pressure $p_h = 80 \text{ cmHg}$, permeate-side pressure $p_l = 20 \text{ cmHg}$

Permeability of *A*: $P_A = 50 \times 10^{-10} \text{ cm}^3 \cdot \text{cm}/(\text{sec} \cdot \text{cm}^2 \cdot \text{cmHg})$

Permeability of *B*: $P_B = 5 \times 10^{-10} \text{ cm}^3 \cdot \text{cm}/(\text{sec} \cdot \text{cm}^2 \cdot \text{cmHg})$

Solution

The script *crfdata.m* specifies membrane data and operating conditions. The function *mArea* defines the function to be integrated to give the membrane area.

```

% crfdata.m: cross-flow calculation data
t = 0.00254; % membrane thickness (cm)
Pm = [50 5]*1e-10; % permeability(cm^3*cm/(s*cm^2*cmHg))
alpa = Pm(1)/Pm(2); % ratio of permeabilities
ph = 80; % feed side pressure(cmHg)
pl = 20; % permeate side pressure(cmHg)
r = pl/ph; % pressure ratio (Plow/Phigh)
qf = 1e4; % feed rate(cm^3/s(STP))
xf = 0.5; % Feed composition (mole fraction)
theta = []; % stage-cut
xr = 0.25; % desired reject composition (mole fraction)

```

The function *crflex* generates the following outputs:

```

>> crflex
ans =
yp: 0.7670
theta: 0.4835
Am: 1.2677e+08
rc: 0.7418
xr: 0.2500

```

PROBLEMS

- 7.1 Methanol (A) is evaporated into a stream of dry air (B) in a cylindrical tube at 328.5 K. The distance from the tube inlet to the liquid surface is $z_2 - z_1 = 0.238$ m. At $T = 328.5$ K, the vapor pressure of methanol is $P_{A0} = 68.4$ kPa and the total pressure is $P = 99.4$ kPa. The binary molecular diffusion coefficient of methanol in air under these conditions is $D_{AB} = 1.991 \times 10^{-5}$ m²/sec. The temperature profile in the tube exhibits a linear characteristic from the liquid surface ($T = 328.5$ K) to the tube inlet ($T = 295$ K). Calculate the molar flux of methanol within the tube and plot the mole fraction profile of methanol from the liquid surface to the flowing air stream.
- 7.2 Gases A and B are diffusing through stagnant gas C at a temperature of 55°C and a pressure of 0.2 atmospheres from point 1 (z_1) to point 2 (z_2). The distance between these two points is 0.001 m. The molar flux of A was measured to be $N_A = 2.115 \times 10^{-5}$ kg mol/(m² · sec). The gas mixture is assumed to be ideal gas. Estimate the molar flux of B (N_B) and plot the mole fraction profile for each component as a function of z .
- Data:* $C_{A1} = 2.229 \times 10^{-4}$, $C_{A2} = 0$, $C_{B0} = 0$, $C_{B2} = 2.701 \times 10^{-3}$, $C_{C1} = 7.208 \times 10^{-3}$, $C_{C2} = 4.730 \times 10^{-3}$, $D_{AB} = 1.47 \times 10^{-4}$, $D_{AC} = 1.075 \times 10^{-4}$, $D_{BC} = 1.245 \times 10^{-4}$.
- 7.3 Dichlorobenzene (A), suspended in stagnant air (B), is sublimed at 25°C and atmospheric pressure. The sublimation is taking place at the surface of a sphere of solid dichlorobenzene with a radius of 3×10^{-3} m. The material balance in the gas phase gives

$$\left(N_A 4\pi r^2 \right)_{\Delta t} = \left(\frac{Vp_A}{RT} \right)_{t+\Delta t} - \left(\frac{Vp_A}{RT} \right)_t \Rightarrow \frac{dp_A}{dt} = \frac{4\pi r^2 k'_c P}{V} \frac{(p_A - p_{A2})}{p_{BM}}$$

The material balance in the spherical particle gives

$$0 = \left(N_A 4\pi r^2 \right)_{\Delta t} + \left(\frac{4}{3} \pi r^3 \frac{\rho_A}{M_A} \right)_{t+\Delta t} - \left(\frac{4}{3} \pi r^3 \frac{\rho_A}{M_A} \right)_t \Rightarrow \frac{dr}{dt} = - \frac{N_A M_A}{\rho_A}$$

where $N_A = (k'_c P/RT)((p_A - p_{A2})/p_{BM})$, $p_{BM} = (p_A - p_{A2})/(\ln(p_A - p_{A2})/(p_A))$, and $k'_c = (D_{AB}/r)$. The vapor pressure of dichlorobenzene at 25°C is 1 mmHg, and the diffusivity in air is 7.39×10^{-6} m²/sec. The density of dichlorobenzene is 1458 kg/m³ and the molecular weight is 147. Calculate the time necessary for the complete sublimation of a single particle of dichlorobenzene if the particle is enclosed in a volume of $V = 0.05$ m³. Plot the partial pressure p_A as a function of the radius r .

- 7.4 Calculate the concentration profile for C_A and determine the effectiveness factor η for a 1st-order irreversible reaction in a cylindrical catalyst particle, where $R = 0.5$ cm, $D_e = 0.1$ cm²/sec, $C_{As} = 0.2$ g mol/cm³, and $k_1 a = 6.4$ sec⁻¹.
- 7.5 A triple-effect forward-feed evaporator is being used to evaporate an organic colloid solution containing 15% solids to a concentrated solution of 60% solids. Saturated steam at 29.82 psia is being used, and the pressure in the vapor phase of the third effect is 1.27 psia. The feed rate is 45,000 lb/hr at 60°F. The heat capacity of the liquid solutions is the same as that of water at the whole concentration range. The overall heat transfer coefficients have been estimated as $U_1 = 530$, $U_2 = 350$, and $U_3 = 195$ Btu/(ft² · hr · °F). If the heat transfer areas of each of the three effects are to be equal, calculate the heat transfer area (ft²) and the amount of steam consumed (lb/hr). The boiling-point elevation of the solutions is assumed to be negligible.
- 7.6 A binary mixture is to be distilled in a distillation column to give a distillate of $x_D = 0.98$ and a bottoms composition of $x_B = 0.01$. The feed composition is $z_F = 0.5$ and the reflux ratio is $R = 2.7$. The feed is a mixture of vapor and liquid and $q = 1$. The equilibrium equation is given by $y = \alpha x/(1 + (1 - \alpha)x)$ with $\alpha = 2.5$. Determine the location of the feed stage and the number of total stages, and plot the equilibrium curve, q -line, and the operating lines as a function of liquid-phase mole fraction.

- 7.7** A 41-stage column with the overhead condenser as stage 1, the feed stage as stage 21, and the reboiler as stage 41 is used to distil a binary mixture. The relative volatility, α , is 2.5. The feed rate is $F = 1$ mol/min, the feed composition is $z_F = 0.5$, and the feed condition is $q = 1$ (bubble-point liquid). The flow rate of the distillate leaving the column is 0.5 mol/min. Plot the steady-state liquid composition profile along the stages for the reflux flow rate of $R = 2.4$, 2.7, and 3.0 mol/min.
- 7.8** A 20-stage column with the overhead condenser as stage 1, the feed stage as stage 10, and the reboiler as stage 20 is used to distil a binary mixture. The relative volatility, α , is 2.5. The feed rate is $F = 1$ mol/min, the feed composition is $z_F = 0.5$, and the feed condition is $q = 1$ (bubble-point liquid). The reflux flow rate is $R = 3$ mol/min and the distillate leaving the column is 0.5 mol/min. The holdups in the condenser and the reboiler are both 5 mol, and the holdup in each stage is maintained constant as 0.5 mol.

At $t = 2$ min, there is 0.2 mol/min step change in the reflux flow rate. Plot the liquid compositions of the distillate and the bottoms for $0 \leq t \leq 30$. Also plot the liquid composition profile along the stages at $t = 30$ min.

- 7.9** A three-component feed mixture of propane(1)/*n*-butane(2)/*n*-pentane(3) is to be distilled in the five-stage distillation column shown in Figure P7.9. The feed composition is $z_{1,3} = 0.3$, $z_{2,3} = 0.3$, $z_{3,3} = 0.4$ (mole fraction) and the feed is supplied to the 3rd stage with the flow rate of $F_j = F_3 = 100$ lb mol/hr. The feed is saturated liquid at 122.33°F and 100 psia. The pressure at each stage is maintained constant as 100 psia. The flow rate of liquid sidecut is $U_1 = L_1/2$ and the flow rate of the bottoms is $L_5 = 50$ lb mol/hr. The specific heat C_{pv}^0 (Btu/lb mol) and the vapor pressure for each component (P_i^s) are given by

$$C_{pv}^0 = a_1 + a_2 T + a_3 T^2 + a_4 T^3 + a_5 T^4 \quad (T : ^\circ\text{F})$$

$$\ln \frac{P_i^s}{P_c} = A_1 - \frac{A_2}{T + A_3} \quad (T : ^\circ\text{F}, P_c : \text{critical pressure})$$

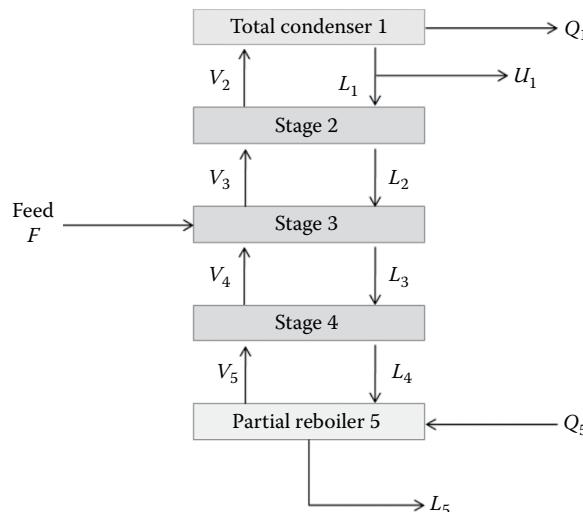


FIGURE P7.9 Five-stage distillation column. (From Henley, E.J. et al., *Separation Process Principles*, 3rd ed., John Wiley & Sons, Inc., Hoboken, NJ, 2011, p. 417.)

TABLE P7.9(1)
Physical Properties of Each Component

Component	P_c (psia)	T_c (R)	ω	A_1	A_2	A_3
Propane $C_3H_8(1)$	617.4	665.9	0.1538	5.35342	3371.084	414.488
<i>n</i> -Butane $C_4H_{10}(2)$	550.7	765.3	0.1954	5.74162	4126.385	409.5179
<i>n</i> -Pentane $C_5H_{12}(3)$	489.5	845.9	0.2387	5.85365	4598.287	394.4148

TABLE P7.9(2)
Parameters of Specific Heat Equations for Each Component

Component	a_1	$a_2 \times 10$	$a_3 \times 10^4$	$a_4 \times 10^7$	$a_5 \times 10^{11}$
Propane $C_3H_8(1)$	15.58683	0.250495	0.140426	-0.352626	1.864467
<i>n</i> -Butane $C_4H_{10}(2)$	20.79783	0.314329	0.192851	-0.458865	2.380972
<i>n</i> -Pentane $C_5H_{12}(3)$	25.64627	0.389176	0.239729	-0.584262	3.079918

The physical properties and parameters for the specific heat equation are shown in Tables P7.9(1) and P7.9(2).⁴⁰ Plot the temperature profile along the stages. Also plot the profiles of liquid and vapor compositions along the stages.⁴⁶

- 7.10 Figure P7.10 shows a distillation column where a feed mixture consisting of methane(1)/ethane(2)/propane(3)/*n*-butane(4)/*n*-pentane(5) is distilled. The number of total stages including the condenser and the reboiler is 13. The feed stream enters to the feed stage (7th stage from the top) at 105°F and 400 psia with the flow rate of $F_j = F_7 = 800$ lb mol/hr and the composition of $z_{1,7} = 0.2$, $z_{2,7} = 0.4625$, $z_{3,7} = 0.3$, $z_{4,7} = 0.03125$, and $z_{5,7} = 0.00625$. The pressure in the column is kept constant as 400 psia and the flow rate of the vapor stream to the first stage (V_1) is 530 lb mol/hr and that of the liquid stream to the first stage (L_1) is 1000 lb mol/hr. The top product is obtained as vapor from the first stage and the flow rate of the liquid sidecut is $U_1 = 0$ lb mol/hr.

The specific heat C_{pv}^0 (Btu/lb mol) and the vapor pressure (P_i^s) are given by

$$C_{pv}^0 = a_1 + a_2 T + a_3 T^2 + a_4 T^3 + a_5 T^4 \quad (T : ^\circ F)$$

$$\ln \frac{P_i^s}{P_c} = A_1 - \frac{A_2}{T + A_3} \quad (T : ^\circ F, P_c : \text{critical pressure})$$

Table P7.10(1) shows the physical properties and Table P7.10(2) shows the parameters of specific heat equations for each component.⁴⁷ Calculate the stage temperatures and vapor and liquid compositions and plot the results versus the stage number.⁴⁸

- 7.11 A membrane is to be used to separate air (oxygen(A) + nitrogen(B)). The feed flow rate is $L_f = 1 \times 10^6$ cm³/s and the feed composition of A is $x_f = 0.209$ (mole fraction). The stage-cut is $\theta = 0.2$ and the ratio of the permeability of oxygen to that of nitrogen is $\alpha = 10$. Use the complete-mixing model to calculate the permeate composition y_p , the reject composition x_R , and the membrane area A_m .⁴⁹

Data: membrane thickness $t = 2.54 \times 10^{-3}$ cm, feed pressure $p_h = 190$ cmHg, permeate-side pressure $p_l = 19$ cmHg

Permeability of A : $P_A = 500 \times 10^{-10}$ cm³ · cm/(sec · cm² · cmHg)

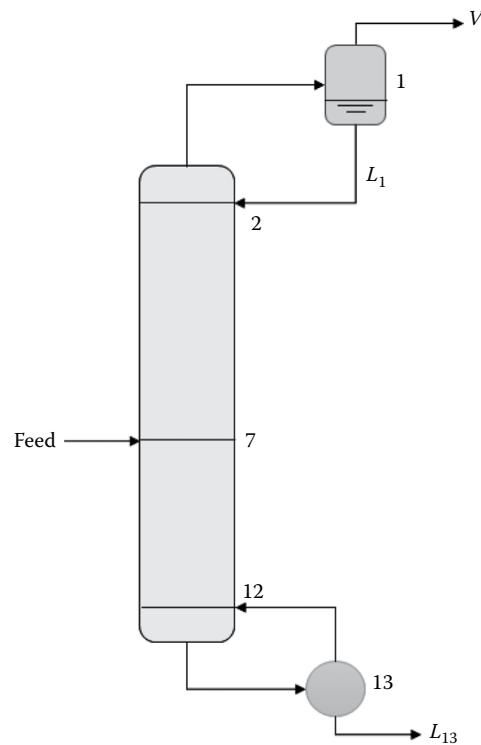


FIGURE P7.10 13-stage distillation column. (From Henley, E.J. et al., *Separation Process Principles*, 3rd ed., John Wiley & Sons, Inc., Hoboken, NJ, 2011, p. 419.)

TABLE P7.10(1)
Physical Properties for Each Component

Component	P_c (psia)	T_c (R)	ω	A_1	A_2	A_3
Methane CH ₄ (1)	673.1	343.9	0.0	5.14135	1742.638	452.974
Ethane C ₂ H ₆ (2)	709.8	550.0	0.1064	5.38389	2847.921	434.898
Propane C ₃ H ₈ (3)	617.4	665.9	0.1538	5.35342	3371.084	414.488
<i>n</i> -Butane C ₄ H ₁₀ (4)	550.7	765.3	0.1954	5.74162	4126.385	409.5179
<i>n</i> -Pentane C ₅ H ₁₂ (5)	489.5	845.9	0.2387	5.853654	4598.287	394.4148

TABLE P7.10(2)
Parameters of Specific Heat Equations for Each Component

Component	a_1	$a_2 \times 10^4$	$a_3 \times 10^4$	$a_4 \times 10^7$	$a_5 \times 10^{11}$
Methane CH ₄ (1)	8.24522	0.038063	0.0886474	-0.074612	0.182296
Ethane C ₂ H ₆ (2)	11.51606	0.140309	0.0854034	-0.110608	0.316220
Propane C ₃ H ₈ (3)	15.58683	0.250495	0.140426	-0.352626	1.864467
<i>n</i> -Butane C ₄ H ₁₀ (4)	20.79783	0.314329	0.192851	-0.458865	2.380972
<i>n</i> -Pentane C ₅ H ₁₂ (5)	25.64627	0.389176	0.239729	-0.584262	3.079918

- 7.12** A membrane is to be used to separate air (oxygen(A)+nitrogen(B)). The feed flow rate is $L_f = 1 \times 10^6 \text{ cm}^3/\text{s}$ and the feed composition of A (oxygen) is $x_f = 0.209$ (mole fraction). The stage-cut is $\theta = 0.2$ and the ratio of the permeability of oxygen to that of nitrogen is $\alpha = 10$. Use the cross-flow model to calculate the permeate composition y_p , the reject composition x_R , and the membrane area A_m .⁴⁵

Data: membrane thickness $t = 2.54 \times 10^{-3} \text{ cm}$, feed pressure $p_h = 190 \text{ cmHg}$, permeate-side pressure $p_l = 19 \text{ cmHg}$

Permeability of A: $P_A = 500 \times 10^{-10} \text{ cm}^3 \cdot \text{cm}/(\text{sec} \cdot \text{cm}^2 \cdot \text{cmHg})$

Permeability of B: $P_B = 50 \times 10^{-10} \text{ cm}^3 \cdot \text{cm}/(\text{sec} \cdot \text{cm}^2 \cdot \text{cmHg})$

REFERENCES

1. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 384, 2008.
2. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice Hall, Boston, MA, p. 418, 2003.
3. Bird, R. B., W. E. Stewart, and E. N. Lightfoot, *Transport Phenomena*, 2nd ed., John Wiley & Sons, Hoboken, NJ, p. 551, 2002.
4. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice Hall, Boston, MA, p. 426, 2003.
5. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 386, 2008.
6. Bird, R. B., W. E. Stewart, and E. N. Lightfoot, *Transport Phenomena*, 2nd ed., John Wiley & Sons, Hoboken, NJ, p. 538, 2002.
7. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 385–386, 2008.
8. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 413, 2008.
9. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice-Hall, Boston, MA, p. 421, 2003.
10. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice-Hall, Boston, MA, p. 482, 2003.
11. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 391, 2008.
12. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 396–398, 2008.
13. Bird, R. B., W. E. Stewart, and E. N. Lightfoot, *Transport Phenomena*, 2nd ed., John Wiley & Sons, Hoboken, NJ, p. 566, 2002.
14. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 402, 2008.
15. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 406–408, 2008.
16. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice-Hall, Boston, MA, p. 417, 2003.
17. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice-Hall, Boston, MA, p. 506, 2003.
18. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 430–431, 2008.
19. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 428–429, 2008.
20. Bird, R. B., W. E. Stewart, and E. N. Lightfoot, *Transport Phenomena*, 2nd ed., John Wiley & Sons, Hoboken, NJ, p. 562, 2002.
21. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 438, 2008.
22. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 439, 2008.

23. Henley, E. J., J. D. Seader, and D. K. Roper, *Separation Process Principles*, 3rd ed., John Wiley & Sons, Inc., Hoboken, NJ, pp. 285–290, 2011.
24. Gilliland, E. R., Estimate of the number of theoretical plates as a function of reflux ratio, *Industrial and Engineering Chemistry*, 32, 1220, 1940.
25. Hengstebeck, R. J., *Distillation*, Reinhold, New York, NY, p. 234, 1961.
26. Liddle, C. J., Improved short-cut method for distillation calculations, *Chemical Engineering*, 75, 137, October 21, 1968.
27. Van-Winkle, M. and W. G. Todd, Optimum fractionation design by simple graphical methods, *Chemical Engineering*, 78, 136, September 20, 1971.
28. Molkavov, Y. K., T. P. Koralina, N. I. Mazurina, and G. A. Nikiforov, An approximation method for calculating the basic parameters of multicomponent fractionation, *International Chemical Engineering*, 12(2), 29–212, 1972.
29. Hohman, E. C. and F. J. Lockhart, Fractional distillation of multicomponent mixtures, *Chemical Technology*, 2, 614, 1972.
30. Eduljee, H. E., Equations replace Gilliland plot, *Hydrocarbon Processing*, 54(9), 120, 1975.
31. Chang, H. Y., Gilliland plot in one equation, *Hydrocarbon Processing*, 64(3), 48, 1985.
32. Harg, K., Equation proposed, *Hydrocarbon Processing*, 64(3), 49, 1985.
33. McCormick, J. E., A correlation for distillation stages and reflux, *Chemical Engineering*, 95, 75, September 26, 1988.
34. Kirkbride, C. G., Process design procedure for multicomponent fractionators, *Petroleum Refinery*, 23(9), 87, 1944.
35. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 516, 1995.
36. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 544–545, 2008.
37. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 551–552, 2008.
38. Wang, J. C. and G. E. Henke, Tridiagonal matrix for distillation, *Hydrocarbon Processing*, 45(8), 155, 1966.
39. Kister, H. Z., *Distillation: Design*, McGraw-Hill, Inc., New York, NY, p. 153, 1992.
40. Henley, E. J. and J. D. Seader, *Equilibrium-Stage Separation Operations in Chemical Engineering*, John Wiley & Sons, Hoboken, NJ, pp. 716–722, 1981.
41. Henley, E. J. and J. D. Seader, *Equilibrium-Stage Separation Operations in Chemical Engineering*, John Wiley & Sons, Hoboken, NJ, p. 568, 1981.
42. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 559–560, 2008.
43. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice Hall, Boston, MA, p. 853, 2003.
44. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice Hall, Boston, MA, pp. 859–860, 2003.
45. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice Hall, Boston, MA, pp. 861–863, 2003.
46. Henley, E. J., J. D. Seader, and D. K. Roper, *Separation Process Principles*, 3rd ed., John Wiley & Sons, Inc., Hoboken, NJ, pp. 417–419, 2011.
47. Henley, E. J. and J. D. Seader, *Equilibrium-Stage Separation Operations in Chemical Engineering*, John Wiley & Sons, Hoboken, NJ, pp. 716–722, 1981.
48. Henley, E. J., J. D. Seader, and D. K. Roper, *Separation Process Principles*, 3rd ed., John Wiley & Sons, Inc., Hoboken, NJ, pp. 418–421, 2011.
49. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice Hall, Boston, MA, p. 857, 2003.



Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

8 Heat Transfer

The transfer of energy in the form of heat occurs in many chemical processes. Heat transfers occur because of a temperature difference, and heat flows from the high temperature to the low temperature region. Heat transfer often occurs in combination with other separation operations such as drying, distillation, and evaporation. Process heat exchange is one of the key operations in the chemical engineering industry. With the rapid rise in the cost of energy, there has been an increasing emphasis in recent years on conservation and efficiency. In order to improve both process and energy efficiency in an operating plant, it is essential to monitor the performance of heat exchange equipment regularly. However, such efforts may result in tedious and repetitive calculations that cannot be abstracted due to the wide-ranging diversity in the kinds of heat transfer equipment currently being employed in the industry. For this reason, a number of modular MATLAB® programs have been developed here.

The main objective of this chapter is to provide readers with the knowledge and skills needed for application of MATLAB to handle heat transfer problems. The MATLAB programs presented in this chapter can be used in undergraduate or graduate courses on heat transfer. Researchers and practicing engineers in the field of chemical engineering can use these MATLAB programs in the analysis and design of heat transfer equipment.

8.1 ONE-DIMENSIONAL HEAT TRANSFER

8.1.1 HEAT TRANSFER IN A ONE-DIMENSIONAL SLAB

Heat transfer in a one-dimensional slab involves conduction, convection, and radiation to the surroundings. Steady-state heat conduction occurring within a one-dimensional slab without heat generation follows Fourier's law given by

$$\frac{q_x}{A} = -k \frac{dT}{dx} \approx -k \frac{\Delta T}{\Delta x}$$

where

q_x (W or J/sec) is the heat transfer rate in the x direction

A (m^2) is the cross-sectional area normal to the direction of heat conduction

k ($\text{W}/\text{m} \cdot \text{K}$) is the thermal conductivity of the solid medium

The convective heat transfer between solids and fluid can be described by

$$\frac{q_x}{A} = h(T_w - T_f)$$

where

h ($\text{W}/\text{m}^2 \cdot \text{K}$) is the heat transfer coefficient

T_w (K) is the surface temperature of the solid

T_f (K) is the fluid temperature

The heat flux at the interface between solids and fluids can be expressed as

$$\left. \frac{q_x}{A} \right|_S = -k \left. \frac{dT}{dx} \right|_S = h(T_w - T_f)$$

where the subscript S denotes the solid surface.

Example 8.1: Heat Transfer in a One-Dimensional Slab¹

Figure 8.1 shows a one-dimensional slab with heat conduction and radiation. One surface of the slab is maintained at temperature T_1 , and the other surface at temperature T_2 has radiative heat transfer with the surroundings that act as a black body at temperature T_a . The radiation from the slab surface can be represented by the Stefan–Boltzmann law:

$$\frac{q_x}{A} \Big|_{x=\Delta x} = \sigma (T_2^4 - T_a^4) \Big|_{x=\Delta x} \quad (\sigma = 5.676 \times 10^{-8} \text{ W/m}^2 \cdot \text{K}^4)$$

Calculate and plot the temperature profile within the slab. What is the corresponding value of T_2 ? The thermal conductivity of the solid slab, k , is dependent upon temperature and is given by $k = 30(1 + 0.002T)$. Assume that the convective heat transfer between the slab and the surroundings is negligible.

Data: $T_1 = 290 \text{ K}$, $T_a = 1273 \text{ K}$, $\Delta x = 0.2 \text{ m}$

Solution

$$q_x = -kA \frac{dT}{dx} = -30(1 + 0.002T) A \frac{dT}{dx} = \sigma (T_2^4 - T_a^4) \Rightarrow \frac{dT}{dx} = -\frac{\sigma (T_2^4 - T_a^4)}{30(1 + 0.002T) A}$$

First an initial estimate for T_2 is provided and solve the differential equation to find a new T_2 . The iterations are continued until the value of T_2 converges. The function *slab1T* defines the differential equation to be solved.

```
function dT = slab1T(x,T,A,T2,Ta,sigma)
% slab1T.m: heat transfer within a one-dimensional slab
dT = - sigma*(T2^4 - Ta^4) / (30*(1 + 0.002*T)*A);
end
```

The script *ht1D* performs the iterative calculations. In the script, k is the number of iterations carried out until the convergence is achieved, and $T(\text{end})$ denotes an updated value of T_2 at each iteration.

```
% ht1D.m: one-dimensional heat transfer (conduction and radiation)
A = 1; T2 = 700; Ta = 1273; sigma = 5.676e-8;
xspan = [0 0.2]; T0 = 290;
deltT = 10; critT = 1e-3; k = 1;
```

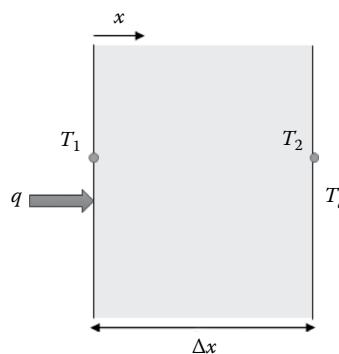


FIGURE 8.1 One-dimensional heat transfer. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, pp. 209–210.)

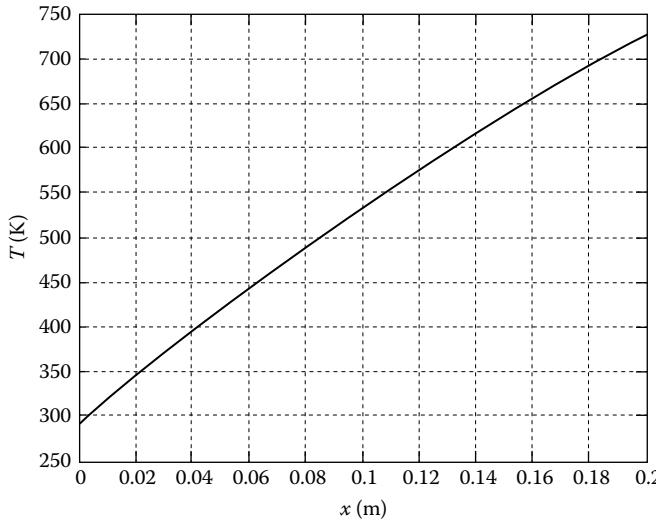


FIGURE 8.2 Heat transfer within one-dimensional slab.

```

while delT > crit
    [x T] = ode45(@slab1T, xspan, T0, [], A,T2,Ta,sigma);
    delT = abs(T2 - T(end));
    T2 = T(end); k = k+1;
end
plot(x,T), xlabel('x(m)'), ylabel('T(K)'), grid
k, T(end)

```

The script *ht1D* generates the following outputs and the plot shown in Figure 8.2:

```

>> ht1D
k =
10
ans =
729.1673

```

8.1.2 HEAT TRANSFER THROUGH A MULTILAYER SLAB

Figure 8.3 shows a multilayer slab of more than one material present. At interfaces between two different solids, the temperature is continuous. The heat flux continuity can be described by

$$T = T_1 \Big|_{x=intf} = T_2 \Big|_{x=intf}, \quad \frac{q_x}{A} = -k_1 \frac{dT_1}{dx} \Big|_{intf} = -k_2 \frac{dT_2}{dx} \Big|_{intf}$$

where the subscript *intf* represents the interface between two different solids.

Since the heat flow *q* is the same in each layer, Fourier's equation for each layer (*A*, *B*, and *C*) can be written as

$$q = \frac{k_A A}{\Delta x_A} (T_1 - T_2) = \frac{k_B A}{\Delta x_B} (T_2 - T_3) = \frac{k_C A}{\Delta x_C} (T_3 - T_4)$$

where *A* is the heat transfer area. Rearrangement of this equation gives

$$\frac{q}{A} = \frac{T_1 - T_4}{\Delta x_A/k_A + \Delta x_B/k_B + \Delta x_C/k_C}$$

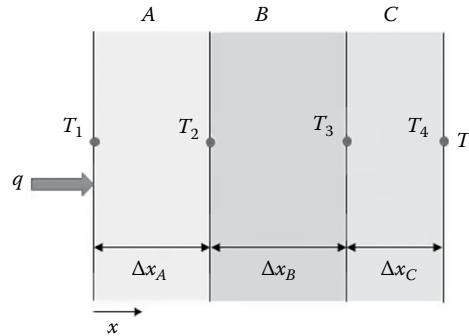


FIGURE 8.3 One-dimensional heat transfer through a multilayered solid slab. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 334.)

8.1.3 HEAT TRANSFER THROUGH MULTILAYER CYLINDERS

Heat transfer often occurs through multilayers of cylinders in the process industries. At steady state, the heat transfer rate q is the same for each layer. From Fourier's law, the heat transfer rate is given by²

$$q = \frac{T_1 - T_5}{\frac{1}{h_i A_i} + \frac{r_1 - r_i}{k_A A_{Alm}} + \frac{r_0 - r_1}{k_B A_{Blm}} + \frac{1}{h_o A_o}}$$

where

r_j ($j = o, 1, i$) represents radius of each cylinder

h_i and h_o (Btu/(hr · ft² · °F)) are heat transfer coefficients inside and outside of the cylinder, respectively

k_A and k_B are heat conductivities (Btu/(hr · ft · °F))

The inner and outer surface areas (ft²) are given by $A_i = 2\pi r_i L$ and $A_o = 2\pi r_o L$, respectively, and the area between the cylinders (ft²) is given by $A_{Alm} = 2\pi r_1 L$. The log-mean area (ft²) is defined by

$$A_{Alm} = \frac{A_o - A_i}{\ln(A_o/A_i)}, \quad A_{Blm} = \frac{A_o - A_i}{\ln(A_o/A_i)}$$

Figure 8.4 shows two concentric hollow cylinders, that is, a pipe with insulation around it.

Example 8.2: Heat Transfer through Multilayer Slab³

Figure 8.5 shows a multilayer slab through which heat is transferred. The thickness of each slab is $L_A = 0.015$ m, $L_B = 0.1$ m, and $L_C = 0.075$ m, and the thermal conductivity of each slab is $k_A = 0.0151$, $k_B = 0.0433$, and $k_C = 0.762$ (W/(m · K)).

1. Calculate the heat flux through the slab if the interior surface is at $T_1 = 255$ K and the exterior surface is at $T_4 = 298$ K.
2. It is proposed to reduce the heat loss by 50% by increasing the thickness of the slab B , L_B . What value of L_B is required?
3. A new slab is to be used instead of the slab B . The thermal conductivity of the new slab is given by $k = 2.5e^{-1225/T}$ (T: K), and the thickness of the new slab is the same as that of the slab B . Calculate the heat flux through the new slab if the interior surface is at $T_1 = 255$ K and the exterior surface is at $T_4 = 298$ K, and plot the temperature profile within the slab. Assume that $q_x/A = -15$ W/m².

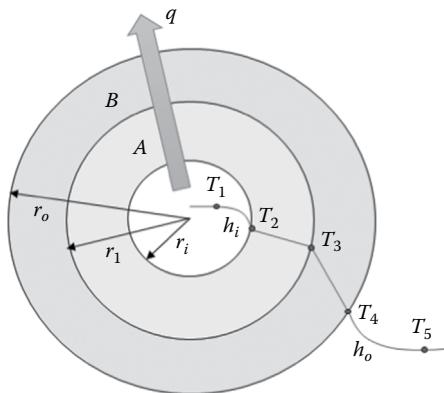


FIGURE 8.4 Heat transfer through multilayer cylinders. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 334.)

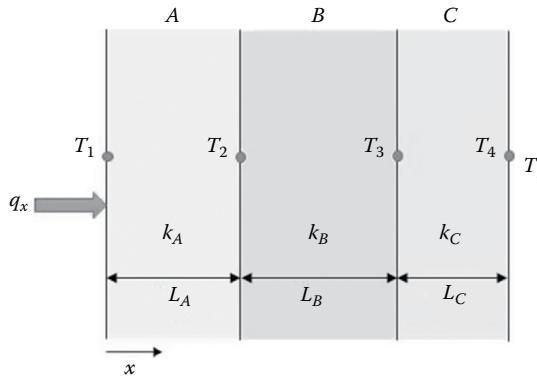


FIGURE 8.5 Heat transfer through a multilayer slab. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 334.)

Solution

1. $q_x/A = (T_1 - T_4)/(\Delta L_A/k_A + \Delta L_B/k_B + \Delta L_C/k_C)$. The following commands calculate the value of q_x/A .

```
>> T1 = 255; T4 = 298; L = [0.015 0.1 0.075]; k = [0.151 0.043 0.762];
>> qx = (T1-T4)/sum(L./k)
qx =
-17.0409
```

We can see that $q_x/A = -17.041 \text{ W/m}^2$.

2. The heat loss per unit area is $q_x/A = -17.0409 \text{ W/m}^2$. Hence, the value of ΔL_B is to be determined such that $(-17.0409)(0.5) = -8.5204 \text{ W/m}^2$. ΔL_B can be directly calculated by

$$\Delta L_B = k_B \left\{ \frac{T_1 - T_4}{q_x/A} - \left(\Delta L_A/k_A + \Delta L_C/k_C \right) \right\}$$

```
>> T1 = 255; T4 = 298; L = [0.015 0.075]; k = [0.151 0.762]; kB = 0.043;
>> dLB = kB*((T1-T4)/(-8.5204) - sum(L./k))
dLB =
0.2085
```

3. According to Fourier's law, one-dimensional heat transfer by conduction for each layer gives

$$\frac{dT}{dx} = \begin{cases} -\frac{q_x/A}{k_A} & : 0 \leq x \leq L_A \\ -\frac{q_x/A}{2.5e^{-1225/T}} & : L_A \leq x \leq L_A + L_B \\ -\frac{q_x/A}{k_C} & : L_A + L_B \leq x \leq L_A + L_B + L_C \end{cases}$$

The function *slabmT* defines these differential equations.

```
function dT = slabmT(x, T, LA, LB, kA, kC, qx)
% slabmT.m: heat transfer through multilayer slab
if x <= LA
    dT = -qx/kA;
elseif x <= LA+LB
    dT = -qx/(2.5*exp(-1225/T));
else
    dT = -qx/kC;
end
```

The total thickness of the multilayered slab is $L_t = L_A + L_B + L_C = 0.19$ m. Thus, the integration interval is $0 \leq x \leq L_t$. The following commands employ the built-in function *ode45* to solve the system of differential equations defined by the function *slabmT* and generate the plot shown in Figure 8.6.

```
>> LA = 0.015; LB = 0.1; LC = 0.075; Lt = LA+LB+LC; kA = 0.151; kC = 0.762;
>> qx = -15; T0 = 255; xspan = [0 Lt];
>> [x T] = ode45(@slabmT, xspan, T0, [], LA, LB, kA, kC, qx);
>> plot(x, T), axis([0 Lt 250 310]), xlabel('x(m)'), ylabel('T(K)')
```

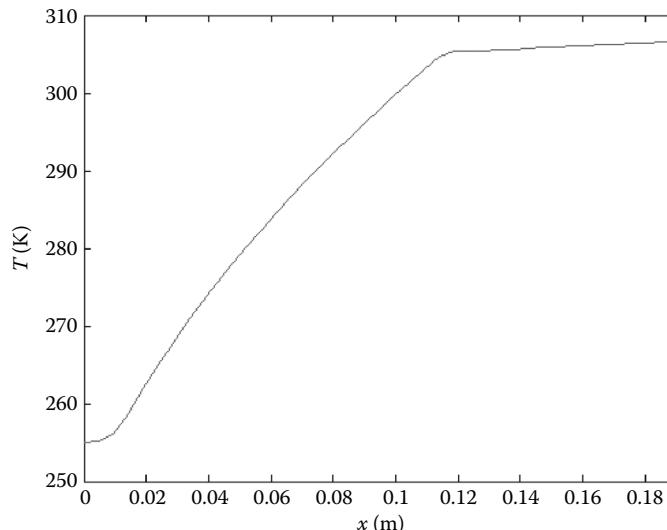


FIGURE 8.6 Temperature profile through a multilayered slab.

8.1.4 HEAT TRANSFER IN A WIRE

The passage of the current in an insulated wire generates heat. Figure 8.7 shows a differential volume of a cylindrical wire of radius R and length L .

An energy balance on a cylinder shell of thickness Δr yields

$$Q_r = -k \frac{dT}{dr}, \quad \frac{d}{dr}(rQ_r) = \frac{I^2}{k_e} r$$

where

Q_r (W/m²) is the heat flux

k (W/(m·K)) is the heat conductivity

I (amps/m²) is current

k_e (Ω^{-1} m⁻¹) is the electrical conductivity

Example 8.3: Heat Transfer in a Wire⁴

An insulated wire is carrying an electrical current. The wire surface is maintained at $T_1 = 15^\circ\text{C}$, and the electrical and thermal conductivities are given by $k = 5 \text{ W/(m·K)}$ and $k_e = 1.4 \times 10^5 e^{0.00357} \Omega^{-1} \text{ m}^{-1}$. The wire radius is $R_1 = 0.004 \text{ m}$, and the total current is maintained at $I_t = 400 \text{ amps}$. Calculate and plot the temperature and heat flux within the wire.

Solution

$$Q_r = -k \frac{dT}{dr}, \quad \frac{d}{dr}(rQ_r) = \frac{I^2}{k_e} r, \quad Q_r = \frac{(rQ_r)}{r}, \quad \text{and} \quad I = \frac{I_t}{\pi R_1^2 / 4}$$

The energy balance gives

$$\frac{dT}{dr} = -\frac{U}{kr}, \quad \frac{dU}{dr} = \frac{I^2}{k_e} r, \quad T(R_1) = T_1, \quad U|_{r=0} = 0$$

where $U = rQ_r$. The function *funQT* defines these differential equations.

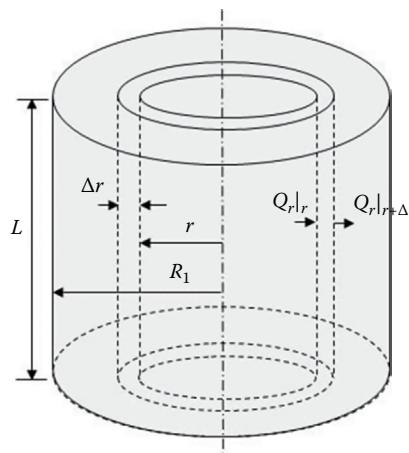


FIGURE 8.7 Differential volume of a wire. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 339.)

```

function dz = funQT(r,z,k,R1,Ir)
if r > 0, Qr = z(2)/r;
else Qr = 0; end
dz(1) = -Qr/k; dz(2) = Ir^2*r/(1.4e5*exp(0.0035*z(1)));
dz = dz';
end

```

The temperature at $r = 0$ is unknown. Thus, an initial guess of T at $r = 0$ is used to solve the system of differential equations to give $T(r = R_1)$. The value of T at $r = 0$ is updated at each iteration until the condition for $T(r = R_1)$ is satisfied. The bisection method may be used in the iterations and the script *bisecQT* performs the iterative calculations.

```

% bisecQT.m: calculation of heat flux and temperature in a wire
clear all;
T1 = 288.15; R1 = 0.004; k = 5; Ir = 400/(pi*R1^2);
T0a = 300; T0b = 400; % guess initial temperature
Teps = 1e-3; Terr = 1e3; iter = 1;
while Terr > Teps
    T0m = (T0a+T0b)/2;
    [r,za] = ode45(@funQT,[0 R1],[T0a 0],[],k,R1,Ir);
    [r,zb] = ode45(@funQT,[0 R1],[T0b 0],[],k,R1,Ir);
    [r,zm] = ode45(@funQT,[0 R1],[T0m 0],[],k,R1,Ir);
    a = za(end,1); b = zb(end,1); m = zm(end,1);
    if (m-T1) > 0, T0b = T0m;
    else T0a = T0m; end
    Terr = abs(T0b - T0a); iter = iter+1;
end
Qr = zm(:,2)./r; T = zm(:,1);
subplot(1, 2), plot(r,Qr), xlabel('r(m)'), ylabel('Qr(W/m^2)'), grid;
subplot(1, 2), plot(r,T), xlabel('r(m)'), ylabel('T(K)'), grid;
fprintf('Number of iterations: %d, Qr at r=R1 (W/m^2): %g, ', iter, Qr(end))
fprintf('T at r=0 (K): %g\n', T(1))

```

The script *bisecQT* produces the following results and the curves shown in Figure 8.8:

```

>> bisecQT
Number of iterations: 18, Qr at r=R1(W/m^2): 276472, T at r=0(K): 389.245

```

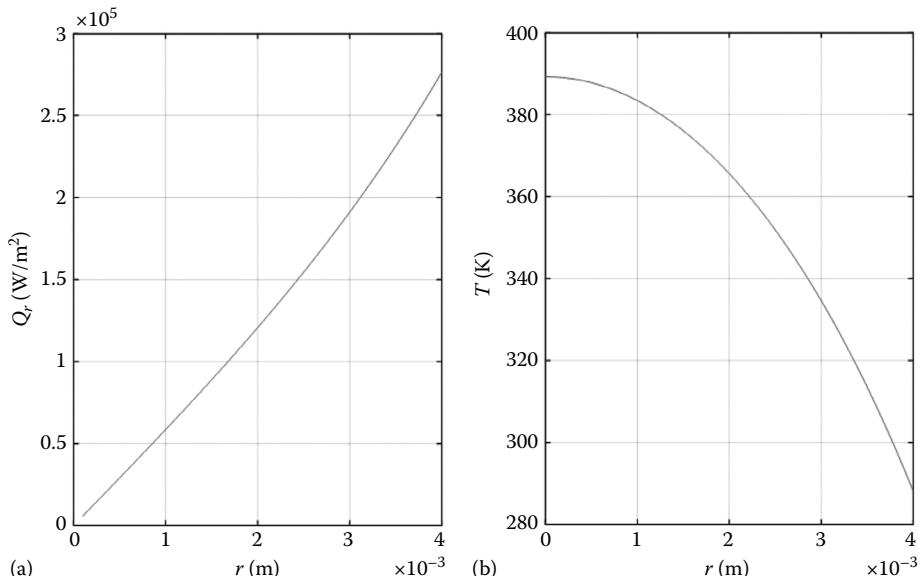


FIGURE 8.8 (a) Heat flux and (b) temperature profile in a wire.

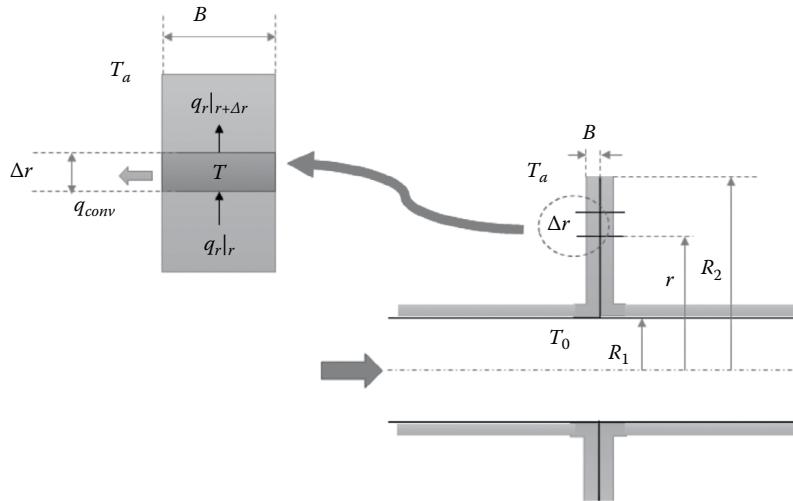


FIGURE 8.9 Cross section of a pipe flange. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, pp. 347–349.)

8.1.5 HEAT LOSS THROUGH PIPE FLANGE⁵

Pipes may be connected by the bolting together of two identical flanges on the pipe ends. Figure 8.9 shows the cross section of pipe flange. The two flanges are symmetrical and only one flange may be considered with heat loss from one exposed circular face and exposed rim as shown in Figure 8.9. The heat transfer rate can be represented as a function of the radius r . At the inner surface where $r = R_1$, the temperature is maintained as the fluid temperature T_0 . The temperature at $r = R_2$ is determined by heat convection from the surface to the ambient temperature T_a .

The heat balance on the different element gives

$$\frac{dT}{dr} = -\frac{q}{k}, \quad \frac{d}{dr}(rq) = -\frac{hr}{B}(T - T_a), \quad q = \frac{(rq)}{r}$$

The initial and boundary conditions are

$$T|_{r=R_1} = T_0, \quad (rq)|_{r=R_2} = R_2 h (T - T_a)|_{r=R_2}$$

The heat flux q_r is given by $q_r = 2\pi r B q$. In order to determine the temperature profile and the heat loss flux q , the initial condition for rq should be determined so that the boundary condition is satisfied. This can be accomplished by defining a nonlinear equation as

$$f = (rq)|_{r=R_2} - R_2 h (T - T_a)|_{r=R_2} = 0$$

The initial condition should satisfy this equation.

Example 8.4: Heat Loss through Pipe Flanges⁶

Consider the union formed by two aluminum flanges. Calculate the total heat loss flux q from a single flange when the average ambient temperature $T_a = 60^\circ\text{F}$. Plot the heat transfer rate q_r versus the radius r . The thermal conductivity of the aluminum pipe and the flange is 133 $\text{Btu}/(\text{hr} \cdot \text{ft} \cdot {}^\circ\text{F})$, the thickness of the flange is 1 in., and $R_1 = 0.0833$ ft and $R_2 = 0.25$ ft. The fluid in the pipe is at $T_0 = 260^\circ\text{F}$ and the heat transfer coefficient to the surroundings is constant at $h = 3 \text{ Btu}/(\text{hr} \cdot \text{ft}^2 \cdot {}^\circ\text{F})$.

Solution

The function *funQL* defines the differential equations.

```
function dy = funQL(r,y,Ta,k,h,B)
q = y(2)/r;
dy(1) = -q/k; dy(2) = -h*r*(y(1)-Ta)/B;
dy = dy';
end
```

The value of the product rq at $r = R_1$ is unknown. Thus, we have to provide an initial guess for $rq|_{r=R_1}$ to solve the differential equations. This procedure continues iteratively until the equation

$$f = (rq)|_{r=R_2} - R_2 h (T - T_a)|_{r=R_2} = 0$$

is satisfied. The bisection method can be used effectively to update the guess. The script *bisecQL* performs these calculations.

```
% bisecQL.m: calculation of heat loss from pipe flange
clear all;
T0 = 260; Ta = 60; R1 = 0.0833; R2 = 0.25; B = 0.5/12; k = 133; h = 3;
qa = 200; qb = 800; % assume initial range for heat loss flux
feps = 1e-3; ferr = 1e-3; iter = 1;
while ferr > feps
    qm = (qa+qb)/2; % midpoint of the assumed temperature range
    [r,ya] = ode45(@funQL,[R1 R2],[T0 qa],[],Ta,k,h,B);
    [r,yb] = ode45(@funQL,[R1 R2],[T0 qb],[],Ta,k,h,B);
    [r,ym] = ode45(@funQL,[R1 R2],[T0 qm],[],Ta,k,h,B);
    fa = ya(end,2) - R2*h*(ya(end,1)-Ta);
    fb = yb(end,2) - R2*h*(yb(end,1)-Ta);
    fm = ym(end,2) - R2*h*(ym(end,1)-Ta);
    if (fa*fm) < 0, qb = qm;
    else qa = qm; end
    ferr = abs(qb - qa); iter = iter+1;
end
Qr = ym(:,2).r; T = ym(:,1); qrate = 2*pi*ym(:,2)*B;
subplot(1, 2)
plot(r,qrate), xlabel('r(ft)'), ylabel('q(Btu/hr)'), grid, axis tight;
subplot(1, 2), plot(r,T), xlabel('r(ft)'), ylabel('T(F)'), grid;
fprintf('Number of iterations: %d, heat transfer rate at r=R2 (Btu/h): %g\n',
    iter, qrate(end))
fprintf('T at r=R2 (K): %g\n', T(end))
```

The script *bisecQL* produces the following outputs and the curves shown in Figure 8.10:

```
>> bisecQL
Number of iterations: 21, heat transfer rate at r=R2 (Btu/h): 38.602
T at r=R2 (K): 256.599
```

8.2 MULTIDIMENSIONAL HEAT CONDUCTION

8.2.1 UNSTEADY-STATE HEAT CONDUCTION

For heat conduction in solids, the velocity terms are zero, and the temperature profile can be represented as a three-dimensional unsteady-state heat conduction equation

$$\rho C_p \frac{\partial T}{\partial t} = k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right)$$

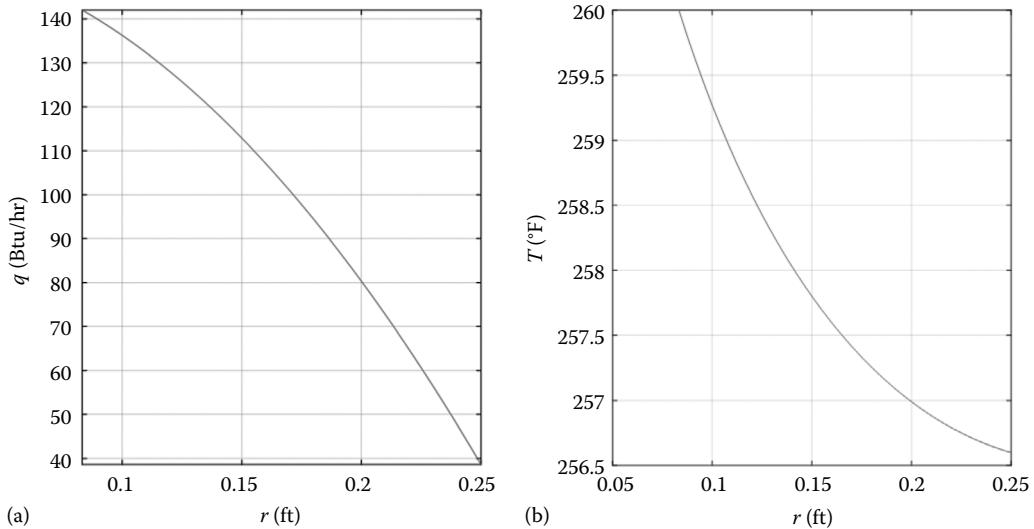


FIGURE 8.10 Profiles of (a) heat transfer rate and (b) temperature.

where C_p is the heat capacity at constant pressure and the heat conductivity k is assumed to be constant within the solid. The two-dimensional unsteady-state heat conduction equation can be expressed as

$$\frac{\partial T}{\partial t} = \alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right)$$

where $\alpha = k/\rho C_p$. The initial and boundary conditions associated with the partial differential equations can be classified into three categories.⁷

Dirichlet conditions (1st kind): The values of the dependent variable are given at fixed values of the independent variable. For example,

$$T = f(x) \quad (t = 0, 0 \leq x \leq 1)$$

or

$$T = T_0 \quad (t = 0, 0 \leq x \leq 1)$$

Boundary conditions can be written as

$$T = f(t) \quad (x = 0, t > 0), \quad T = T_1 \quad (x = 1, t > 0)$$

Neumann conditions (2nd kind): The derivative of the dependent variable is given as a constant or as a function of the independent variable. For example,

$$\frac{\partial T}{\partial x} = 0 \quad (x = 1, t \geq 0)$$

Cauchy conditions: A combination of Dirichlet and Neumann conditions is classified as Cauchy conditions.

Robbins conditions (3rd kind): The derivative of the dependent variable is given as a function of the dependent variable itself. For example,

$$k \frac{\partial T}{\partial x} = h(T - T_f) \quad (x = 0, t \geq 0)$$

8.2.2 METHOD OF LINES⁸

In the method of lines, only the spatial derivatives are discretized using finite differences except the time derivatives. For the one-dimensional parabolic equation $\partial u / \partial t = \alpha(\partial^2 u / \partial x^2)$, the discretization of only the spatial derivative term yields

$$\frac{du_i}{dt} = \frac{\alpha}{\Delta x^2} (u_{i+1} - 2u_i + u_{i-1})$$

The complete set of ordinary differential equations for $0 \leq i \leq N$ can be written as

$$\begin{aligned} \frac{du_1}{dt} &= \frac{\alpha}{\Delta x^2} (u_2 - 2u_1 + u_0) \\ &\vdots \\ \frac{du_i}{dt} &= \frac{\alpha}{\Delta x^2} (u_{i+1} - 2u_i + u_{i-1}) \\ &\vdots \\ \frac{du_{N-1}}{dt} &= \frac{\alpha}{\Delta x^2} (u_N - 2u_{N-1} + u_{N-2}) \end{aligned}$$

The equations at the boundary

$$\frac{du_0}{dt} = \frac{\alpha}{\Delta x^2} (u_1 - 2u_0 + u_{-1}), \quad \frac{du_N}{dt} = \frac{\alpha}{\Delta x^2} (u_{N+1} - 2u_N + u_{N-1})$$

are specified according to the boundary conditions. For example, if a Dirichlet condition is given, $u_0 = \beta$ (constant) and

$$\frac{du_0}{dt} = 0, \quad u_0(0) = \beta$$

If a Neumann condition is given at this boundary, that is, $\frac{du}{dx} \Big|_{0,t} = 0$ at $x = 0$, the partial derivative is replaced by a central difference approximation $\frac{du}{dx} \Big|_{0,t} = (u_1 - u_{-1}) / 2\Delta x = 0$ and we have

$$\frac{du_0}{dt} = \frac{\alpha}{\Delta x^2} (2u_1 - 2u_0)$$

The function *parab1D* solves the unsteady-state one-dimensional parabolic partial differential equation. The function *parab2D* solves the unsteady-state two-dimensional parabolic partial differential equation. When a Dirichlet condition is given at the boundary for the one-dimensional parabolic

partial differential equation, the function *parabDbc* can also be used to solve the problem. The basic syntax of the functions *parab1D*, *parab2D*, and *parabDbc* are as follows:

```
function [x,t,u] = parab1D(nx, nt, dx, dt, alpha, u0, bc, func, varargin)
function [u r] = parabDbc(nx, nt, dx, dt, alpha, u0, bci, bcf)
function [x,y,t,u] = parab2D(nx,ny,nt,dx,dy,dt,alpha,u0,bc,func,varargin)
```

where *u* is the matrix of dependent variables (*u*(*x*,*t*)); *nx* is the number of divisions in the *x* direction; *ny* is the number of divisions in the *y* direction; *nt* is the number of divisions in the *t* direction; *dx* and *dy* are the distances between grid points in *x* and *y* directions, respectively; *dt* is the time step; and *u0* is the distribution vector for *u* at *t* = 0.

In the function *parab1D*, *bc* is the matrix containing types and values of boundary conditions in the *x* direction as its elements (2×2 or 2×3 matrix). The first and second rows of *bc* represent conditions at lower *x* and at upper *x*, respectively. The first column of *bc* specifies the type of the boundary conditions (1: Dirichlet condition, values of *u* are specified in the second column; 2: Neumann condition, values of *u'* are specified in the second column; 3: Robbins condition, the constants and coefficients for *u* are specified in the second and third columns). *bci* and *bcf* are values of boundary conditions at each end of *x*.

In the function *parab2D*, *bc* is the matrix containing types and values of boundary conditions in the *x* and *y* directions as its elements (4×2 or 4×3 matrix). The order of the boundary conditions is lower *x*, upper *x*, lower *y*, and upper *y* (these are stored in the first four row of the matrix *bc*). The first row of *bc* specifies the type of the boundary conditions (1: Dirichlet condition, values of *u* are specified in the second column; 2: Neumann condition, values of *u'* are specified in the second column; 3: Robbins condition, the constants and coefficients for *u* are specified in the second and third columns).

parab1D.m

```
function [x,t,u] = parab1D(nx,nt,dx,dt,alpha,u0,bc,func,varargin)
% Solve 1-dimensional parabolic PDE
% Outputs:
%     x, t; vectors of x and t values
%     u: matrix of dependent variables [u(x,t)]
% Inputs:
%     nx, nt: number of divisions in x and t direction
%     dx, dy: x and t increment
%     alpha: coefficient of equation
%     u0: vector of u distribution at t=0
%     bc: a matrix containing types and values of boundary conditions
%         in x direction (2x2 or 2x3 matrix)
%         row 1: conditions at lower x, row 2: conditions at upper x
%         1st column: type of condition
%         (1: Dirichlet condition, values of u in 2nd column
%         2: Neumann condition, values of u' in 2nd column
%         3: Robbins condition, 2-3 columns contain constant and
%             coef. of u)

% Initialization
if nargin < 7
    error('Insufficient number of inputs.')
end

nx = fix(nx); x = [0:nx]*dx; nt = fix(nt);
t = [0:nt]*dt; r = alpha*dt/dx^2;
u0 = (u0(:).'); % confirm column vector
if length(u0) ~= nx+1
    error('Incorrect length of the initial condition vector.')
end
```

```

[a,b]=size(bc);
if a ~= 2
    error('Invalid number of boundary conditions.')
end
if b < 2 | b > 3
    error('Invalid boundary condition.')
end
if b == 2 & max(bc(:,1)) <= 2
    bc = [bc zeros(1, 2)];
end

u(:,1) = u0; c = zeros(nx+1,1);
% Iteration according to t
for n = 2:nt+1 % lower x boundary condition
    switch bc(1)
        case 1
            A(1) = 1; c(1) = bc(1, 2);
        case {2, 3}
            A(1) = -3/(2*dx) - bc(1, 3);
    A(1, 2) = 2/dx; A(1, 3) = -1/(2*dx); c(1) = bc(1, 2);
    end
    % Interior points
    for i = 2:nx
        A(i,i-1) = -r; A(i,i) = 2*(1+r); A(i,i+1) = -r;
        c(i) = r*u(i-1,n-1) + 2*(1-r)*u(i,n-1) + r*u(i+1,n-1);
    if nargin >= 8 % Nonhomogeneous equation
            intercept = feval(func,0,x(i),t(n),varargin{:});
            slope = feval(func,1,x(i),t(n),varargin{:}) - intercept;
            A(i,i) = A(i,i) - dt*slope;
            c(i) = c(i) + dt*feval(func,u(i,n-1),x(i),t(n-1),...
                varargin{:}) + dt*intercept;
        end
    end
    switch bc(1, 2) % upper x boundary condition
    case 1
        A(nx+1,nx+1) = 1; c(nx+1) = bc(2);
    case {2, 3}
        A(nx+1,nx+1) = 3/(2*dx) - bc(2, 3);
    A(nx+1,nx) = -2/dx; A(nx+1,nx-1) = 1/(2*dx); c(nx+1) = bc(2);
    end
    u(:,n) = inv(A)*c;
end

```

parab2D.m

```

function [x,y,t,u] = parab2D(nx,ny,nt,dx,dy,dt,alpha, ...
    u0,bc,func,varargin)
% Solve 2-dimensional parabolic PDE: use explicit formula
% Outputs:
%     x, y, t: vectors of x, y and t values
%     u: 3D array of dependent variables [u(x,y,t)]
% Inputs:
%     nx, ny, nt: number of divisions in x, y and t direction
%     dx, dy: x and y increments
%     dt: t increments (leave empty to use the default values)
%     alpha: coefficient of equation
%     u0: matrix of u distribution at t=0 [u0(x,y)]

```

```

% bc: a matrix containing types and values of boundary conditions
%      in x and y directions (4x2 or 4x3 matrix)
%      order of appearing: lower x, upper x, lower y, upper y
%      in rows 1 to 4 of the matrix bc
% 1st column: type of condition
%      (1: Dirichlet condition, values of u in 2nd column
%      2: Neumann condition, values of u' in 2nd column
%      3: Robbins condition, 2-3 columns contain constant and
%          coef. of u)

% Initialization
if nargin < 9
    error(' Invalid number of inputs.')
end

nx = fix(nx); x = [0:nx]*dx; ny = fix(ny); y = [0:ny]*dy;
% Check dt for stability
tmax = dt*nt;
if isempty(dt) | dt > (dx^2+dy^2)/(16*alpha)
    dt = (dx^2+dy^2)/(16*alpha); nt = tmax/dt+1;
fprintf('\ndt is adjusted to %6.2e (nt=%3d)\n',dt,fix(nt))
end
nt = fix(nt); t = [0:nt]*dt; rx = alpha*dt/dx^2; ry = alpha*dt/dy^2;
[r0,c0] = size(u0);
if r0 ~= nx+1 | c0 ~= ny+1
    error('Incorrect size of the initial condition matrix.')
end

[a,b]=size(bc);
if a ~= 4
    error('Invalid number of boundary conditions.')
end
if b < 2 | b > 3
    error('Invalid boundary condition.')
end
if b == 2 & max(bc(:,1)) <= 2
    bc = [bc zeros(1, 4)];
end

% Solution of PDE
u(:,:,1) = u0;
for n = 1:nt
    for i = 2:nx
        for j = 2:ny
            u(i,j,n+1) = rx*(u(i+1,j,n)+u(i-1,j,n))+ ry*(u(i,j+1,n)...
                + u(i,j-1,n)) + (1-2*rx-2*ry)*u(i,j,n);
            if nargin >= 10
                u(i,j,n+1) = u(i,j,n+1) +...
                    dt*feval(func,u(i,j,n),x(i),y(j),t(n),varargin{:});
            end
        end
    end
% Lower x boundary condition
switch bc(1)
case 1
    u(1,2:ny,n+1) = bc(1, 2) * ones(1,ny-1,1);

```

```

case {2, 3}
    u(1,2:ny,n+1) = (-2*bc(1,2)*dx + 4*u(2,2:ny,n+1) ...
        - u(3,2:ny,n+1)) / (2*bc(1,3)*dx + 3);
end

% Upper x boundary condition
switch bc(1, 2)
case 1
    u(nx+1,2:ny,n+1) = bc(2) * ones(1,ny-1,1);
case {2, 3}
    u(nx+1,2:ny,n+1) = (-2*bc(2,2)*dx - 4*u(nx,2:ny,n+1) ...
        + u(nx-1,2:ny,n+1)) / (2*bc(2,3)*dx - 3);
end

% Lower y boundary condition
switch bc(1, 3)
case 1
    u(2:nx,1,n+1) = bc(2, 3) * ones(nx-1,1,1);
case {2, 3}
    u(2:nx,1,n+1) = (-2*bc(3,2)*dy + 4*u(2:nx,2,n+1) ...
        - u(2:nx,3,n+1)) / (2*bc(3,3)*dy + 3);
end

% Upper y boundary conditionCorner nodes
switch bc(1, 4)
case 1
    u(2:nx,ny+1,n+1) = bc(2, 4) * ones(nx-1,1,1);
case {2, 3}
    u(2:nx,ny+1,n+1) = (-2*bc(4,2)*dy - 4*u(2:nx,ny,n+1) ...
        + u(2:nx,ny-1,n+1)) / (2*bc(4,3)*dy - 3);
end
end

% Corner nodes
u(1,1,:) = (u(1,2,:) + u(2,1,:)) / 2;
u(nx+1,1,:) = (u(nx+1,2,:) + u(nx,1,:)) / 2;
u(1,ny+1,:) = (u(1,ny,:) + u(2,ny+1,:)) / 2;
u(nx+1,ny+1,:) = (u(nx+1,ny,:) + u(nx,ny+1,:)) / 2;

```

parabDbc.m

```

function [u r] = parabDbc(nx,nt,dx,dt,alpha,u0,bci,bcf)
% Solve 1-dimensional parabolic PDE (constant Dirichlet boundary
% conditions)
% Outputs:
%     u: matrix of dependent variables [u(x,t)]
% Inputs:
%     nx, nt: number of divisions in x and t direction
%     dx, dy: x and t increments
%     alpha: coefficient of equation
%     u0: row vector of u distribution at t=0
%     bci, bcf: boundary conditions at both ends of x (scalar)

r = alpha*dt/dx^2; A = zeros(nx-1,nx-1);
u = zeros(nt+1,nx+1); u(:,1) = bci*ones(nt+1,1);
u(:,nx+1) = bcf*ones(nt+1,1);
u(1,:) = u0; A(1) = 1 + 2*r; A(1, 2) = -r;

```

```

for i = 2:nx-2
    A(i,i) = 1 + 2*r; A(i,i-1) = -r; A(i,i+1) = -r;
end
A(nx-1,nx-2) = -r; A(nx-1,nx-1) = 1 + 2*r;
b(1) = u0(2) + u0(1)*r;
for i = 2:nx-2
    b(i,1) = u0(i+1);
end
b(nx-1,1) = u0(nx) + u0(nx+1)*r;
[L U] = lu(A);
for j = 2:nt+1
    y = L\b; x = U\y; u(j,2:nx) = x'; b = x;
    b(1) = b(1) + bci*r; b(nx-1,1) = b(nx-1,1) + bcf*r;
end
end

```

Example 8.5: One-Dimensional Parabolic PDE for Heat Transfer

A wall made of brick is 0.5 m thick, and the temperature of the wall is 100°C at $t = 0$. The thermal diffusivity of the brick is $\alpha = 4.52 \times 10^{-7}$ m/sec². The temperature on the both sides of the wall is suddenly dropped to 18°C. Calculate and plot the temperature profile within the brick wall during 5 hr (18,000 sec) with the interval of 300 sec. Assume that the number of nodes in the x direction is 25.

Solution

Since the values of the temperature at both sides of the wall are given at fixed values, the boundary conditions are Dirichlet conditions and the function *parabDbc* can be used to solve the problem. The following commands produce desired outputs and the temperature profile shown in [Figure 8.11](#):

```

>> alpha = 4.52e-7; nx = 25; dx = 0.5/nx; dt = 300; nt = 18000/dt;
>> u0 = 100*ones(1,nx+1); bci = 18; bcf = 18;
>> [u r] = parabDbc(nx, nt, dx, dt, alpha, u0, bci, bcf);
>> surf(u), axis([0 nx+1 0 nt+1 0 110])
>> view([-217 30]), xlabel('x'), ylabel('t'), zlabel('T')
>> r
r =
0.3390

```

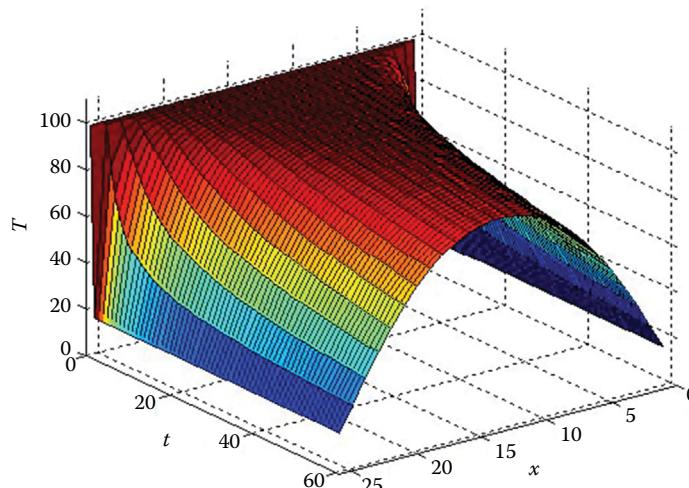


FIGURE 8.11 Temperature profile within the wall.

8.2.3 STEADY-STATE HEAT CONDUCTION

The steady-state heat conduction within the three-dimensional solid may be expressed by the elliptic partial differential equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = 0$$

The two-dimensional steady-state heat conduction problem can be described by

$$\nabla^2 T = 0 \quad (\text{Laplace equation})$$

$$\nabla^2 T = f(x, y) \quad (\text{Poisson equation})$$

$$\nabla^2 T + g(x, y)T = f(x, y) \quad (\text{Helmholtz equation})$$

where $\nabla^2 u = (\partial^2 u / \partial x^2) + (\partial^2 u / \partial y^2)$. Introducing the central difference approximation to the Laplace equation $\nabla^2 u = (\partial^2 u / \partial x^2) + (\partial^2 u / \partial y^2) = 0$, we have

$$\frac{1}{\Delta x^2} (u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) + \frac{1}{\Delta y^2} (u_{i,j+1} - 2u_{i,j} + u_{i,j-1}) = 0$$

Rearrangement of this equation yields

$$-2 \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} \right) u_{i,j} + \left(\frac{1}{\Delta x^2} \right) u_{i+1,j} + \left(\frac{1}{\Delta x^2} \right) u_{i-1,j} + \left(\frac{1}{\Delta y^2} \right) u_{i,j+1} + \left(\frac{1}{\Delta y^2} \right) u_{i,j-1} = 0$$

The function *ellipticPDE* solves the elliptic partial differential equation using this equation.

```
function [x,y,U] = ellipticPDE(nx,ny,dx,dy,bc,f)
% Solve 2-dimensional elliptic PDE
% output:
%   x, y: vectors of x and y values
%   U: matrix of dependent variables (U(x,y))
% input:
%   nx, ny: number of divisions in x and y direction
%   dx, dy: x and y increments
%   f: constant for Poisson equation
%   bc: a matrix containing types and values of boundary conditions
%       in x and y directions (4x2 or 4x3 matrix)
%       order of appearing: lower x, upper x, lower y, upper y
%       in rows 1 to 4 of the matrix bc
%       1st column: type of condition:
%           (1: Dirichlet condition, values of u in 2nd column
%           2: Neumann condition, values of u' in 2nd column
%           3: Robbins condition, 2-3 columns contain constant and coef. of u)
%
% Initialization
if nargin < 5
    error('Invalid number of inputs.')
end
```

```

[a,b] = size(bc);
if a ~= 4
    error('Invalid number of boundary conditions.')
end
if b < 2 | b > 3
    error('Invalid boundary condition.')
end
if b == 2 & max(bc(:,1)) <= 2
    bc = [bc zeros(1, 4)];
end

if nargin < 6 | isempty(f)
    f = 0;
end

nx = fix(nx); x = [0:nx]*dx; ny = fix(ny); y = [0:ny]*dy;
dx2 = 1/dx^2; dy2 = 1/dy^2;

% Coefficient matrix and constant vector
n = (nx+1)*(ny+1); A = zeros(n); c = zeros(n,1);
onex = diag(diag(ones(nx-1))); oney = diag(diag(ones(ny-1)));

% Interior nodes
i = [2:nx];
for j = 2:ny
    ind = (j-1)*(nx+1)+i;
    A(ind,ind) = -2*(dx2+dy2)*onex;
    A(ind,ind+1) = A(ind,ind+1) + dx2*onex;
    A(ind,ind-1) = A(ind,ind-1) + dx2*onex;
    A(ind,ind+nx+1) = A(ind,ind+nx+1) + dy2*onex;
    A(ind,ind-nx-1) = A(ind,ind-nx-1) + dy2*onex;
    c(ind) = f*ones(nx-1,1);
end

% Lower x boundary condition
switch bc(1)
case 1
    ind = ([2:ny]-1)*(nx+1)+1;
    A(ind,ind) = A(ind,ind) + oney;
    c(ind) = bc(1, 2)*ones(ny-1,1);
case {2, 3}
    ind = ([2:ny]-1)*(nx+1)+1;
    A(ind,ind) = A(ind,ind) - (3/(2*dx) + bc(1, 3))*oney;
    A(ind,ind+1) = A(ind,ind+1) + 2/dx*oney;
    A(ind,ind+2) = A(ind,ind+2) - 1/(2*dx)*oney;
    c(ind) = bc(1, 2)*ones(ny-1,1);
end

% Upper x boundary condition
switch bc(1, 2)
case 1
    ind = [2:ny]*(nx+1);
    A(ind,ind) = A(ind,ind) + oney;
    c(ind) = bc(2)*ones(ny-1,1);
case {2, 3}
    ind = [2:ny]*(nx+1);
    A(ind,ind) = A(ind,ind) + (3/(2*dx) - bc(2, 3))*oney;
    A(ind,ind-1) = A(ind,ind-1) - 2/dx*oney;
end

```

```

A(ind,ind-2) = A(ind,ind-2) + 1/(2*dx)*oney;
c(ind) = bc(2)*ones(ny-1,1);
end

% Lower y boundary condition
switch bc(1, 3)
case 1
    ind = [2:nx];
A(ind,ind) = A(ind,ind) + onex;
c(ind) = bc(2, 3)*ones(nx-1,1);
case {2, 3}
    ind = [2:nx];
A(ind,ind) = A(ind,ind) - (3/(2*dy) + bc(3))*onex;
A(ind,ind+nx+1) = 2/dy*onex;
A(ind,ind+2*(nx+1)) = -1/(2*dy)*onex;
c(ind) = bc(2, 3)*ones(nx-1,1);
end

% Upper y boundary condition
switch bc(1, 4)
case 1
    ind = ny*(nx+1)+[2:nx];
A(ind,ind) = A(ind,ind) + onex;
c(ind) = bc(2, 4)*ones(nx-1,1);
case {2, 3}
    ind = ny*(nx+1)+[2:nx];
A(ind,ind) = A(ind,ind) + (3/(2*dy) - bc(3, 4))*onex;
A(ind,ind-(nx+1)) = A(ind,ind-(nx+1)) - 2/dy*onex;
A(ind,ind-2*(nx+1)) = A(ind,ind-2*(nx+1)) + 1/(2*dy)*onex;
c(ind) = bc(2, 4)*ones(nx-1,1);
end

% Corner nodes
A(1) = 1; A(1, 2) = -1/2; A(1,nx+2) = -1/2; c(1) = 0;
A(nx+1,nx+1) = 1; A(nx+1,nx) = -1/2; A(nx+1,2*(nx+1)) = -1/2;
c(nx+1) = 0;

A(ny*(nx+1)+1,ny*(nx+1)+1) = 1;
A(ny*(nx+1)+1,ny*(nx+1)+2) = -1/2;
A(ny*(nx+1)+1,(ny-1)*(nx+1)+1) = -1/2;
c(ny*(nx+1)+1) = 0;

A(n,n) = 1; A(n,n-1) = -1/2; A(n,n-(nx+1)) = -1/2; c(n) = 0;
u = inv(A)*c; % solve equation

% Rearrange results in matrix form
for k = 1:ny+1
    U(k,1:nx+1) = u((k-1)*(nx+1)+1:k*(nx+1))';
end

```

Example 8.6: Two-Dimensional Elliptic Equations for Heat Transfer⁹

The temperature profile in a two-dimensional thin metal plate can be described by the two-dimensional elliptic partial differential equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f$$

where f is assumed to be constant. The metal plate is made of an alloy that has a melting point of 800°C and a thermal conductivity of 16 W/(m·K). The plate is subject to an electric current that creates a uniform heat source within the plate. The amount of heat generated is $Q' = 100 \text{ kW/m}^3$. All four edges of the plate are in contact with a fluid at 25°C. The set of Robbins boundary conditions is

$$\begin{aligned}\left.\frac{\partial T}{\partial x}\right|_{0,y} &= 5\{T(0,y) - 25\}, \quad \left.\frac{\partial T}{\partial x}\right|_{1,y} = 5\{25 - T(1,y)\}, \quad \left.\frac{\partial T}{\partial y}\right|_{x,0} = 5\{T(x,0) - 25\}, \\ \left.\frac{\partial T}{\partial y}\right|_{x,1} &= 5\{25 - T(x,1)\}\end{aligned}$$

Plot the temperature profiles within the plate.

In order to solve the set of equations, all the values of dependent variables have to be rearranged as a column vector and numbered. The finite difference approximation for this problem has the form

$$-2\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)u_n + \left(\frac{1}{\Delta x^2}\right)u_{n+1} + \left(\frac{1}{\Delta x^2}\right)u_{n-1} + \left(\frac{1}{\Delta y^2}\right)u_{n+p+1} + \left(\frac{1}{\Delta y^2}\right)u_{n-p-1} = f$$

When the Laplace equation is being solved, $f = 0$. For the Poisson equation, the value of f is assumed constant throughout the plate.

If the boundary condition is of the Dirichlet type, $u_N = \text{(constant)}$. However, if the boundary condition is of the Neumann or Robbins type, forward or backward difference is used to evaluate the 1st-order derivative at the boundaries.

$x = 0$: forward difference (N is a node on the line $x = 0$)

$$\left.\frac{\partial u}{\partial x}\right|_{x=0} = \frac{1}{2\Delta x}(-3u_N + 4u_{N+1} - u_{N+2})$$

$x = L$: backward difference (N is a node on the line $x = L$)

$$\left.\frac{\partial u}{\partial x}\right|_{x=L} = \frac{1}{2\Delta x}(3u_N - 4u_{N-1} + u_{N-2})$$

$y = 0$: forward difference (N is a node on the line $y = 0$)

$$\left.\frac{\partial u}{\partial y}\right|_{y=0} = \frac{1}{2\Delta y}(-3u_N + 4u_{N+p+1} - u_{N+2p+2})$$

$y = L$: backward difference (N is a node on the line $y = L$)

$$\left.\frac{\partial u}{\partial y}\right|_{y=L} = \frac{1}{2\Delta y}(3u_N - 4u_{N-p-1} + u_{N-2p-2})$$

Solution

The script *tempPDE* specifies the dimension of the plate and boundary conditions. This script calls the function *ellipticPDE* to solve the partial differential equation.

```
% tempPDE.m
% Solve Laplace/Poisson equations using finite difference method

clear all;
bcond = ['Lower x boundary condition:' 
          'Upper x boundary condition:' 
          'Lower y boundary condition:' 
          'Upper y boundary condition:'];
```

```

intercal = 1;
while intercal
    distx = input('Length of the plate in x direction (m): ');
    disty = input('Width of the plate in y direction (m): ');
    ndx = input('Number of divisions in x direction: ');
    ndy = input('Number of divisions in y direction: ');
    rhf = input('Right-hand side of the equation: ');
    disp('Boundary conditions:')
    for k = 1:4
        disp(bcond(k,:))
    disp('1: Dirichlet, 2: Neumann, 3: Robbins')
    bc(k,1) = input('Select boundary condition: ');
    if bc(k,1) < 3
        bc(k,2) = input('Value = ');
    else
        disp(' u" = (beta) + (gamma)*u')
    bc(k,2) = input('Constant (beta): ');
    bc(k,3) = input('Coefficient (gamma): ');
    end
    end
    [x,y,T] = ellipticPDE(ndx,ndy,distx/ndx,disty/ndy,bc,rhf);
    surf(y,x,T), xlabel('x(m)'), ylabel('y(m)'), zlabel('T(deg C)')
    colorbar, view(135,45), disp(' ')
    intercal = input('Repeat calculations (no:0, yes:1)? '); clc
end

```

The script *tempPDE* can be executed as follows. The temperature profile is obtained as shown in Figure 8.12.

```

>> tempPDE
Length of the plate in x direction (m): 1
Width of the plate in y direction (m): 1
Number of divisions in x direction: 20

```

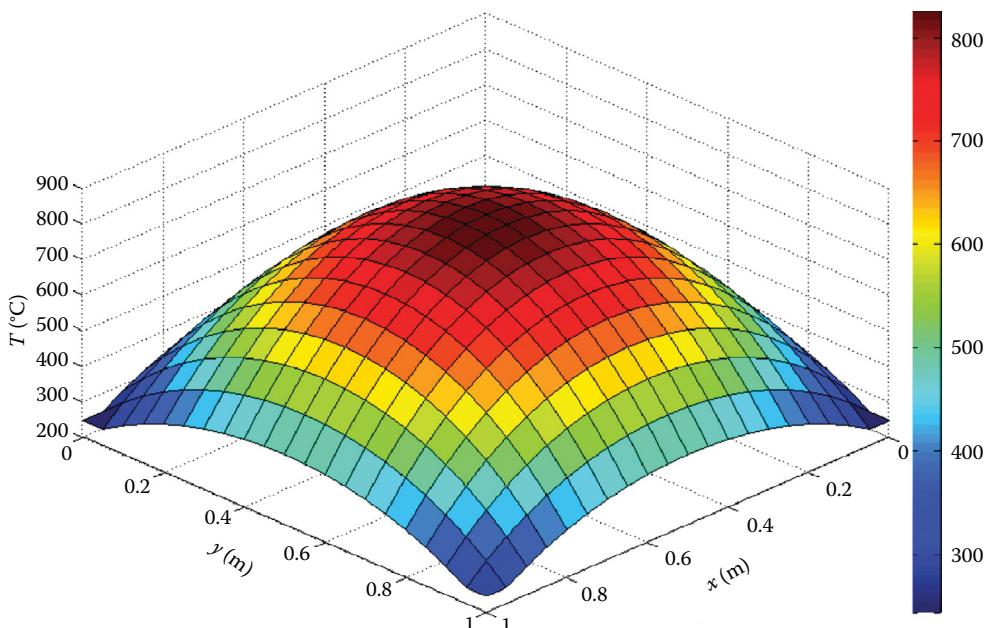


FIGURE 8.12 Temperature profile within a metal plate.

```

Number of divisions in y direction: 20
Right-hand side of the equation: -100e3/16
Boundary conditions:
Lower x boundary condition:
1: Dirichlet, 2: Neumann, 3: Robbins
Select boundary condition: 3
u' = (beta) + (gamma)*u
Constant (beta): -5*25
Coefficient (gamma): 5
Upper x boundary condition:
1: Dirichlet, 2: Neumann, 3: Robbins
Select boundary condition: 3
u' = (beta) + (gamma)*u
Constant (beta): 5*25
Coefficient (gamma): -5
Lower y boundary condition:
1: Dirichlet, 2: Neumann, 3: Robbins
Select boundary condition: 3
u' = (beta) + (gamma)*u
Constant (beta): -5*25
Coefficient (gamma): 5
Upper y boundary condition:
1: Dirichlet, 2: Neumann, 3: Robbins
Select boundary condition: 3
u' = (beta) + (gamma)*u
Constant (beta): 5*25
Coefficient (gamma): -5
Repeat calculations (no:0, yes:1)? 0

```

8.3 HEAT EXCHANGER

8.3.1 SHELL-AND-TUBE HEAT EXCHANGER WITHOUT PHASE CHANGE

8.3.1.1 Log-Mean Temperature Difference

In the design of the countercurrent shell-and-tube heat exchanger, it is important to minimize the number of shells. For a simple countercurrent shell-and-tube heat exchanger shown in Figure 8.13, the minimum number of shells and the log-mean temperature difference can be given as follows¹⁰:

$$\Delta t_1 = T_1 - t_2, \quad \Delta t_2 = T_2 - t_1, \quad P = \frac{t_2 - t_1}{T_1 - t_1}, \quad R = \frac{T_1 - T_2}{t_2 - t_1}$$

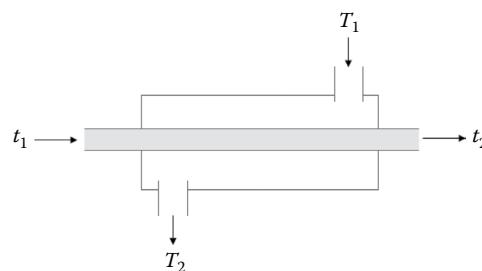


FIGURE 8.13 A simple shell-and-tube heat exchanger.

1. $R \neq 1$:

$$P' = \frac{1 - \left(\frac{PR-1}{P-1} \right)^{1/N}}{R - \left(\frac{PR-1}{P-1} \right)^{1/N}}, \quad F = \frac{\frac{\sqrt{R^2+1}}{R-1} \log \left(\frac{1-P'}{1-RP'} \right)}{\log A}, \quad A = \frac{\frac{2}{P'} - 1 - R + \sqrt{R^2+1}}{\frac{2}{P'} - 1 - R - \sqrt{R^2+1}}$$

2. $R = 1$:

$$P'' = \frac{P}{N - P(N-1)}, \quad F = \frac{\left(\frac{\sqrt{R^2+1} \cdot P''}{\ln 10(1-P'')} \right)}{\log B}, \quad B = \frac{\frac{2}{P''} - 2 + \sqrt{2}}{\frac{2}{P''} - 2 - \sqrt{2}}$$

$$\Delta T_{lm} = \frac{\Delta t_1 - \Delta t_2}{\ln \left(\frac{\Delta t_1}{\Delta t_2} \right)}$$

where

Δt_1 is the larger terminal temperature difference

Δt_2 is the smaller terminal temperature difference

N is the number of shells required

ΔT_{lm} is the log-mean temperature difference

The updated log-mean temperature difference is given by

$$\Delta T_{lm} = F \cdot \Delta T_{lm}$$

where F is the correction factor.

The updated log-mean temperature difference is evaluated using the following procedure:

1. Input required data and calculate P and R .
2. Set $N = 1$ and compare the value of R with 1.
3. If $R \neq 1$, calculate P' and A :
 - i. If $A > 0$, calculate F and go to step 5.
 - ii. If $A < 0$, set $N = N + 1$ and go to step 3.
4. If $R = 1$, calculate P'' and B :
 - i. If $B > 0$, calculate F and go to step 5.
 - ii. If $B < 0$, set $N = N + 1$ and go to step 3.
5. If $F \leq 0.75$, set $N = N + 1$ and go to step 3.
6. If $\Delta t_1 = \Delta t_2$, $\Delta T_{lm} = \Delta t_1$.
7. If $\Delta t_1 \neq \Delta t_2$, calculate ΔT_{lm} .
8. Update ΔT_{lm} .

This procedure is executed by the script *shellLMTD*.

```
% shellLMTD.m : number of required shells and LMTD
clear all;
% Data
```

```

T1 = input('Hot fluid inlet temperature (deg.F): ');
T2 = input('Hot fluid outlet temperature (deg.F): ');
t1 = input('Cold fluid inlet temperature (deg.F): ');
t2 = input('Cold fluid outlet temperature (deg.F): ');

N = 1; % Initialize N
Dt1 = T2 - t1; Dt2 = T1 - t2;
% Compute P and R
P = (t2 - t1)/(T1 - T2); R = (T1 - T2)/(t2 - t1);
% Test F and R
A = -1; B = -1; F = 0.1; % Initial F
while (F <= 0.75)
    if ( R > 1 || R < 1)
        while (A < 0)
            Pp = (1 - ((P*R-1)/(P-1))^(1/N)) / (R-((P*R-1)/(P-1))^(1/N));
A = (2/Pp - 1 - R + sqrt(R^2 +1)) / (2/Pp - 1 - R - sqrt(R^2 +1));
if (A < 0)
    N = N + 1;
end
end
F = ( sqrt(R^2 +1) * log10((1-Pp)/(1-Pp*R)) / (R-1) ) / (log10(A));
else % R = 1
    while (B < 0)
        Ppp = P / (N-P*(N-1));
B = (2/Ppp - 2 + sqrt(2)) / (2/Ppp - 2 - sqrt(2));
if (B < 0)
    N = N + 1;
end
end
F = ( sqrt(R^2 +1) * Ppp / (log(10*(1-Ppp))) ) / (log10(B));
end
if (F <= 0.75)
    N = N + 1;
end
end
% Compute LMTD
if (Dt1 == Dt2)
    LMTD = Dt1;
else
    LMTD = (Dt1-Dt2)/(log(Dt1/Dt2));
end
cLMTD = F*LMTD; % Corrected LMTD
% Output
fprintf('\nNumber of shells = %3d\n', N);
fprintf('F factor = %9.4f\n', F);
fprintf('Corrected LMTD = %9.4f\n', cLMTD);

```

Example 8.7: Number of Shells and Log-Mean Temperature Difference

A hot fluid is cooled by cooling water in a countercurrent shell-and-tube heat exchanger. Determine the number of shells required, the correction factor F , and the updated log-mean temperature difference (LMTD).

Data: hot fluid inlet temperature (T_1) = 250°F, hot fluid outlet temperature (T_2) = 100°F, cooling water inlet temperature (t_1) = 80°F, cooling water outlet temperature (t_2) = 120°F.

Solution

The script *shellLMTD* produces the following results:

```
>> shellLMTD
Hot fluid inlet temperature (deg.F): 250
Hot fluid outlet temperature (deg.F): 100
Cold fluid inlet temperature (deg.F): 80
Cold fluid outlet temperature (deg.F): 120

Number of shells = 2
F factor = 0.9189
Corrected LMTD = 54.0029
```

8.3.2 TUBE-SIDE HEAT TRANSFER COEFFICIENT

The tube-side Reynolds number is defined as

$$N_{Rei} = \frac{d_i G_i}{\mu_i}$$

where the mass flow rate G_i is obtained from

$$G_i = \frac{4W_i}{n\pi d_i^2}$$

where

W_i is the total mass flow rate of the tube-side fluid

n is the number of tubes per pass

The Graetz number G_{zi} for the tube-side fluid can be defined as

$$G_{zi} = \frac{d_i^2 G_i C_{pi}}{k_i L}$$

The Nusselt number, $N_{ui} = h_i d_i / k_i$, is specified according to the tube-side Reynolds number as

$$N_{Re} \leq 2100 \text{ (laminar flow)}: N_{ui} = \begin{cases} 3.66 + \frac{0.085 G_{zi}}{1 + 0.047 G_{zi}^{2/3}} \left(\frac{\mu_i}{\mu_o} \right)^{0.14} & : G_{zi} \leq 100 \\ 1.86 G_{zi}^{1/3} \left(\frac{\mu_i}{\mu_o} \right)^{0.14} & : G_{zi} > 100 \end{cases}$$

$$2100 < N_{Rei} < 10^4: N_{ui} = 0.116 \left(N_{Rei}^{2/3} - 125 \right) P_{ri} \left[1 + \left(\frac{d_i}{L} \right)^{2/3} \right] \left(\frac{\mu_i}{\mu_o} \right)^{0.14}$$

$$N_{Rei} \geq 10^4: N_{ui} = 0.023 N_{Rei}^{0.8} P_{ri}^{0.33} \left(\frac{\mu_i}{\mu_o} \right)^{0.14}$$

where the Prandtl number P_{ri} is defined as

$$P_{ri} = \frac{C_{pi} \mu_i}{k_i}$$

8.3.3 SHELL-SIDE HEAT TRANSFER COEFFICIENT¹¹

The local heat transfer coefficient on the shell side is calculated using the following procedure:

1. Calculate the shell-side Reynolds number

$$N_{Res} = \frac{d_0 W_0}{\mu_0 S_m}$$

where S_m is the cross-flow area at or near the center line for one cross-flow section given by¹²

$$S_m = \begin{cases} l_b \left[D_s - d_{otl} + \frac{d_{otl} - d_0}{p_n} (p_T - d_0) \right] & \text{: square layouts} \\ l_b \left[D_s - d_{otl} + \frac{d_{otl} - d_0}{p_T} (p_T - d_0) \right] & \text{: triangular layouts} \end{cases}$$

where

l_b is the baffle spacing

D_s is the shell inside diameter

p_T is the tube pitch

d_{otl} is the shell outer tube limit

The tube pitch normal to flow, p_n , is defined as p_T for in-line square layout, $p_T/\sqrt{2}$ for rotated square layout, and $p_T/2$ for triangular layout.

2. Calculate the heat transfer coefficient for an ideal tube bank from¹³

$$\frac{h_o}{C_{po} G_o} = j_{H,s} P_{ro}^{-2/3} \left(\frac{\mu_o}{\mu_{0\omega}} \right)^{0.14}, \quad j_{H,s} = a_1 N_{Res}^{a2} \left(\frac{1.33 d_o}{p_T} \right)^a, \quad a = \frac{a_3}{1 + 0.14 N_{Res}^{a4}}$$

Table 8.1 shows the correlational coefficients a_i ($i = 1, 2, 3, 4$) for various flow configurations.

3. Determine the correction factor ϕ_c for baffle configuration effects from

$$\phi_c = F_{tc} + 0.54 (1 - F_{tc})^{0.345}$$

where F_{tc} is the fraction of total tubes in cross-flow and is given by

$$F_{tc} = \frac{1}{\pi} \left[\pi + 2 \left(\frac{D_s - 2l_c}{d_{otl}} \right) \sin \left(\cos^{-1} \frac{D_s - 2l_c}{d_{otl}} \right) - 2 \cos^{-1} \frac{D_s - 2l_c}{d_{otl}} \right]$$

where l_c is the baffle cut in dimensions of length.

4. Calculate the correction factor ϕ_l for baffle leakage effects using

$$\phi_l = \alpha + (1 - \alpha) \exp \left(-2.2 \frac{S_{tb} + S_{sb}}{S_m} \right)$$

$$\alpha = 0.44 \left(1 - \frac{S_{sb}}{S_{tb} + S_{sb}} \right), \quad S_{tb} = 15.81 d_o N_T (1 + F_{tc}), \quad S_{sb} = \frac{D_s \delta_{sb}}{2} \left[\pi - \cos^{-1} \left(1 - \frac{2l_c}{D_s} \right) \right]$$

where N_T is the total number of tubes in the bundle.

TABLE 8.1
Correlational Coefficients for Various Flow Configurations

Tube Arrangement	Range of N_{Res}	a_1	a_2	a_3	a_4
Triangular	$10^5 - 10^4$	0.321	-0.388	1.450	0.519
	$10^4 - 10^3$	0.321	-0.388		
	$10^3 - 10^2$	0.593	-0.477		
	$10^2 - 10$	1.360	-0.657		
	<10	1.400	-0.667		
Square (in-line)	$10^5 - 10^4$	0.370	-0.395	1.187	0.370
	$10^4 - 10^3$	0.107	-0.266		
	$10^3 - 10^2$	0.408	-0.460		
	$10^2 - 10$	0.900	-0.631		
	<10	0.970	-0.667		
Square (rotated)	$10^5 - 10^4$	0.370	-0.396	1.930	0.500
	$10^4 - 10^3$	0.370	-0.396		
	$10^3 - 10^2$	0.730	-0.500		
	$10^2 - 10$	0.498	-0.656		
	<10	1.550	-0.667		

Source: Serth, R.W., *Process Heat Transfer*, Academic Press, Elsevier, Burlington, MA, 2007, p. 249.

5. Determine the correction factor ϕ_b for bundle bypassing effects from

$$\phi_b = \exp \left[-C_{bh} F_{bp} \left\{ 1 - 2 \left(\frac{N_{ss}}{N_c} \right)^{1/3} \right\} \right]$$

$$F_{bp} = \frac{D_s - d_{out}}{S_m} l_b, \quad N_c = \frac{D_s}{p_p} \left(1 - 2 \frac{l_c}{D_s} \right), \quad C_{bh} = \begin{cases} 1.35 : N_{Res} \leq 100 \\ 1.25 : N_{Res} > 100 \end{cases}$$

where the tube pitch parallel to flow, p_p , is defined as p_T for in-line square layout, $p_T/\sqrt{2}$ for rotated square layout, and $\sqrt{3}p_T/2$ for triangular layout. $\phi_b = 1$ when $N_{sc}/N_c \geq 0.5$.

6. Calculate the correction factor ϕ_r for adverse temperature gradient buildup at low Reynolds numbers by using

$$\phi_r = \begin{cases} 1 : & N_{Res} > 100 \\ 1 - \left(1 - \phi_r^* \right) \left(1.24 - 0.0124 N_{Res} \right) : & 20 \leq N_{Res} \leq 100 \\ \phi_r^* : & N_{Res} < 20 \end{cases}$$

$$\phi_r^* = 3.76 \left[\ln \left(N_c + N_{co} \right) \right]^{-0.15 N_b^{-0.174}}, \quad N_{co} = \frac{0.81 l_c}{p_p}, \quad N_b = \frac{L}{l_b} - 1$$

7. Calculate the correction factor due to unequal baffle spacing at inlet and outlet:

$$\phi_s = \frac{\left(N_b - 1 \right) + l_{in}^{*(1-n)} + l_{out}^{*(1-n)}}{\left(N_b - 1 \right) + l_{in}^* + l_{out}^*}, \quad l^* = \frac{l_c}{l_b}$$

The exponent n takes on values of 0.333 for $N_{Res} \leq 100$ and 0.6 otherwise.

8. The shell-side heat transfer coefficient is given by

$$h_o = h_{o,ideal} \phi_c \phi_l \phi_b \phi_r \phi_s$$

8.3.4 PRESSURE DROP IN THE TUBE SIDE

The pressure drop for the tube-side fluid is calculated by

$$\Delta P_i = \frac{2f_i u_i^2 \rho_i L N_{tp}}{d_i (\mu_i / \mu_\infty)^{0.14}} + K \left(\frac{u_i^2}{2g} \right) \rho_i N_{tp}$$

The pressure losses in the inlet and exit nozzles can be estimated from

$$\Delta P_{nozzle} = 1.5 \rho_i \left(\frac{u_{nozzle}^2}{2} \right)$$

8.3.5 PRESSURE DROP IN THE SHELL SIDE¹³

1. Calculate the ideal tube bank friction factor from

$$f_s = b_1 N_{Res}^{b_2} \left(\frac{1.33 d_o}{p_T} \right)^b, \quad b = \frac{b_3}{1 + 0.14 N_{Res}^{b_4}}$$

The correlation constants b_i ($i = 1, 2, 3, 4$) are shown in Table 8.2 for various tube configurations.

2. Calculate the ideal tube bank pressure drop for one cross-flow path:

$$\Delta P_b = 2 f_s N_c \frac{G_o^2}{\rho_o} \left(\frac{\mu_o}{\mu_\infty} \right)^{-0.14}$$

TABLE 8.2
Correlation Constants for Various Tube Configurations

Tube Arrangement	Range of N_{Res}	b_1	b_2	b_3	b_4
Triangular	$10^5 - 10^4$	0.372	-0.123	7.00	0.500
	$10^4 - 10^3$	0.486	-0.152		
	$10^3 - 10^2$	4.570	-0.476		
	$10^2 - 10$	45.100	-0.973		
	<10	48.000	-1.000		
Square (in-line)	$10^5 - 10^4$	0.391	-0.148	6.30	0.378
	$10^4 - 10^3$	0.0815	0.022		
	$10^3 - 10^2$	6.090	-0.602		
	$10^2 - 10$	32.100	-0.963		
	<10	35.000	-1.000		
Square (rotated)	$10^5 - 10^4$	0.303	-0.126	6.59	0.520
	$10^4 - 10^3$	0.333	-0.136		
	$10^3 - 10^2$	3.500	-0.476		
	$10^2 - 10$	26.200	-0.913		
	<10	32.000	-1.000		

Source: Serth, R.W., *Process Heat Transfer*, Academic Press, Elsevier, Burlington, MA, 2007, p. 249.

3. Calculate the pressure drop for an ideal window section from

$$\Delta P_o = \begin{cases} (2 + 0.6N_{co}) \frac{W_o^2}{2\sqrt{S_m S_{co}} \rho_o} : & N_{Res} > 100 \\ 0.026 \frac{W_o^2 \mu_0}{\sqrt{S_m S_{co}} \rho_o} \left(\frac{N_{co}}{p_T - d_o} + \frac{l_b}{d_{co}^2} \right) + \frac{W_o^2}{\sqrt{S_m S_{co}} \rho_o} : & N_{Res} \leq 100 \end{cases}$$

$$S_{co} = \frac{D_s^2}{4} \left[\cos^{-1} \left(1 - \frac{2l_c}{D_s} \right) - \left(1 - \frac{2l_c}{D_s} \right) \sqrt{1 - \left(1 - \frac{2l_c}{D_s} \right)^2} \right] - \frac{N_T}{8} (1 - F_{tc}) \pi d_o^2$$

$$d_{co} = \frac{4S_{co}}{(\pi/2)N_T (1 - F_{tc}) d_o + 2D_s \cos^{-1} (1 - 2l_c/D_s)}$$

4. Determine the correction factor R_l for the effect of baffle leakage on the pressure drop¹⁵:

$$R_l = \exp \left[-1.33b \left(\frac{S_{tb} + S_{sb}}{S_m} \right)^m \right], \quad m = -0.15b + 0.8, \quad b = 1 + \frac{S_{sb}}{S_{tb} + S_{sb}}$$

5. Calculate the correction factor R_b for the bundle bypassing effect on the pressure drop:

$$R_b = \exp \left[-C_{bp} F_{bp} \left\{ 1 - 2 \left(\frac{N_{ss}}{N_c} \right)^{1/3} \right\} \right], \quad C_{bp} = \begin{cases} 4.5 : N_{Res} \leq 100 \\ 3.7 : N_{Res} > 100 \end{cases}$$

6. Determine the correction factor for unequal baffle spacing at the inlet and outlet by

$$R_s = l_{in}^{*(2-n)} + l_{out}^{*(2-n)}, \quad n = \begin{cases} 1 & : N_{Res} \leq 100 \\ 0.2 & : N_{Res} > 100 \end{cases}$$

7. The shell-side pressure drop is given by

$$\Delta P_o = R_b \Delta P_b \left[(N_b - 1) R_l + \left(1 + \frac{N_{co}}{N_c} \right) R_s \right] + N_b R_l \Delta P_o$$

8.3.6 HEAT EXCHANGER CALCULATION PROCEDURE

1. Input the required data: shell inside diameter (D_s), baffle spacing (l_b), tube outer diameter (d_o), tube inner diameter (d_i), tube layout (triangular, square, rotated square), tube pitch (p_T), tube length (L), thickness of the tube sheet (l_s), number of tube passes (N_{Tp}), total number of tubes (N_T), baffle cut (l_c), number of sealing strips (N_{ss}).
2. Specify two reference temperatures (T_{ref1} , T_{ref2}) and physical properties at these temperatures: densities, viscosities, thermal conductivities, and heat capacities of the shell- and tube-side fluids and the thermal conductivity of the tube material (k_o).
3. Input the operating conditions: the inlet and outlet temperatures of the shell-side fluid and tube-side fluid, flow rates of shell-side and tube-side flows, and fouling coefficients.
4. If a temperature or a flow rate is unknown, determine the unknown quantity from an energy balance equation. If there are two unknown quantities, assume one unknown quantity and calculate the other one by energy balances.

5. Calculate the tube-side heat transfer coefficient using an appropriate correlation. For a start, assume that the viscosity correction factor, $(\mu/\mu_{i_0})^{0.14}$, is equal to unity.
6. Calculate the shell-side heat transfer coefficient.
7. Estimate the tube wall temperature: $t_w = t_b + (h_o/(h_i + h_o))(T_b - t_b)$.
8. Calculate the viscosity correction factor $(\mu/\mu_w)^{0.14}$.
9. Repeat steps 5–8 until two successive values of t_w agree within a preset tolerance.
10. Calculate the tube wall coefficient: $h_w = 2k_w/(d_o - d_i)$.
11. Calculate the overall heat transfer coefficient.
12. Calculate the log-mean temperature difference ΔT_{lm} and the correction factor $F = F_T$ for multiple tube passes.
13. If the heat duty is known, calculate the required heat transfer area $A_{req} = Q/UF_T\Delta T_{lm}$. Find the difference between the actual heat transfer area and the calculated value and go to step 15.
14. If the heat duty is unknown, estimate the heat duty from $Q = U\Delta T_{lm}F_T A$. Using the estimated Q , calculate iteratively the unknown quantity assumed in step 4.
15. Determine the tube-side and shell-side pressure drop and stop the calculation procedure.

Various MATLAB functions perform these calculations steps. In the calculation of physical properties, two reference temperatures in the operation range are specified and physical properties at these temperatures are used to find physical properties at a certain temperature T . For example,

$$\text{Liquid viscosity: } \mu = ae^{b/T}, \quad \text{Vapor viscosity: } \rho = \rho_{ref} \frac{T_{ref}}{T}$$

$$y = a + bT \quad (y = \mu, k, C_p, \rho)$$

The constants a and b are determined from two reference values. The function *hxvis* calculates the tube-side and shell-side viscosities. The function *hxthc* calculates heat conductivities, the function *hxcp* calculates heat capacities, and the function *hxrho* finds densities.

hxvis.m

```
function mu = hxvis(muref,Tref,T,fstate)
% Calculate viscosity at T using two reference temperatures and viscosities
% input:
% muref: viscosity vector at reference temperature vector Tref
% Tref: reference temperature vector
% T: temperature (K) at which viscosity is to be determined
% fstate: state of fluid (1: liquid(mu = A*exp(B/T), 2: vapor(mu = A+BT))
% output:
% mu: viscosity (Ns/m^2)
if fstate == 1 % liquid
    mu = muref(1)*exp(Tref(2)*(Tref(1) - T)/(T*Tref(1) - ...
    Tref(2))*log(muref(2)/muref(1)));
else % vapor
    mu = (muref(2)*Tref(1) - muref(1)*Tref(2) + ...
    T*(muref(1)-muref(2)))/(Tref(1) - Tref(2));
end
end
```

hxthc.m

```
function xk = hxthc(xkref,Tref,T)
% Calculate heat conductivity at T using two ref. temperatures and
% conductivities
% input:
```

```
% xkref: heat conductivity vector at ref. temperature vector Tref
% Tref: reference temperature vector (K)
% T: temperature (K) at which heat conductivity is to be determined
% output:
% xk: heat conductivity (W/m/K) (xk = A+B*T)
xk = (xkref(2)*Tref(1) - xkref(1)*Tref(2) +...
T*(xkref(1)-xkref(2)))/(Tref(1) - Tref(2));
end


function Cp = hxcp(cpref,Tref,T)
function Cp = hxcp(cpref,Tref,T)
% Calculate heat capacity at T using two ref. temperatures and heat
capacities
% input:
% cpref: heat capacity vector (J/kg/K) at ref. temperature vector Tref
% Tref: reference temperature vector (K)
% T: temperature (K) at which heat capacity is to be determined
% output:
% Cp: heat capacity (J/kg/K) (Cp = A+B*T)
Cp = (cpref(2)*Tref(1) - cpref(1)*Tref(2) +...
T*(cpref(1)-cpref(2)))/(Tref(1) - Tref(2));
end
```

```
hxrho.m
function rho = hxrho(rhoref,Tref,T,fstate)
% Calculate density at T using two ref. temperatures and densities
% input:
% rhoref: density vector (kg/m^3) at ref. temperature vector Tref
% Tref: reference temperature vector (K)
% T: temperature (K) at which density is to be determined
% fstate: state of fluid (1: liquid(rho = A+B*T, 2: vapor(rho = A/T))
% output:
% rho: density (kg/m^3)
if fstate == 1 % liquid
    rho = (rhoref(2)*Tref(1) - rhoref(1)*Tref(2) +...
    T*(rhoref(1)-rhoref(2)))/(Tref(1) - Tref(2));
else % vapor
    rho = rhoref(1)*Tref(1)/T;
end
end
```

The heat transfer coefficients for the tube side and shell side are calculated by the functions *htctube* and *htcshell*, respectively.

```
htctube.m
function Htube = htctube(Nre,Pr,D,L,Xk,Phi)
% Calculate convective heat transfer coefficient within tube
% input:
% Nre: Reynolds number
% Pr: Prandtl number
% D: inside diameter of tube (mm)
% L: tube length (m)
% Xk: thermal conductivity of fluid (W/m/K)
% Phi: viscosity correction factor
% output:
```

```

% Htube: convective heat transfer coefficient within tube
Dm = D*1e-3; % mm->m
if Nre <= 2100
    Gw = Nre*Pr*Dm/L;
    if Gw > 100
        Nu = 1.86*Phi*Gw^0.333;
    else
        Nu = 3.66 + 0.085*Gw*Phi/(1 + 0.047*Gw^0.6667);
    end
elseif Nre < 1e4
    Nu = 0.116*(Nre^0.6667 - 125)*Pr^0.333 * (1+(Dm/L)^0.6667)*Phi;
else
    Nu = 0.023*Phi*(Nre^0.8)*(Pr^0.333);
end
Htube = Nu*Xk/Dm;
end

htcshell.m
function Hshell = htcshell(D,Ds,L,Lbc,Lbin,Lbout,Lc,Dotl,Dsb, ...
Pt,Nt,Nss,w,Visc,Cp,Xk,Phi,Layout)
% Calculate shell-side heat transfer coefficient
% input:
% D: outside diameter of tube (mm)
% Ds: inside diameter of shell (mm)
% L: tube length (m)
% Lbc: central baffle spacing (mm)
% Lbin: inlet baffle spacing (mm)
% Lbout: outlet baffle spacing (mm)
% Lc: baffle cut (mm)
% Dotl: shell outer tube limit (mm)
% Dsb: shell-baffle clearance (mm)
% Pt: tube pitch (mm)
% Nt: total number of tubes in the bundle
% Nss: number of pairs of sealing strips
% w: flow rate in shell (kg/s)
% Visc: shell-side fluid viscosity (Ns/m^2)
% Cf: heat capacity of shell-side fluid (J/kg/K)
% Xk: heat conductivity of shell-side fluid (W/m/K)
% Phi: viscosity correction factor
% Layout: tube layout (1:triangular, 2:in-line square, 3:rotated square)
% output:
% Hshell: shell-side heat transfer coefficient
% Correlational coefficients for tube arrangement
a1 = [0.321 0.321 0.593 1.360 1.400; 0.370 0.107 0.408 0.900 0.970;
      0.370 0.370 0.730 0.498 1.550];
a2 = -[0.388 0.388 0.477 0.657 0.667; 0.395 0.266 0.460 0.631 0.667;
      0.396 0.396 0.500 0.656 0.667];
a3 = [1.450 1.187 1.930];
a4 = [0.519 0.370 0.500];
switch Layout
    case 1
        Pp = 0.866*Pt; Pn = Pt/2; Pd = Pt;
    case 2
        Pp = Pt; Pn = Pt; Pd = Pn;
    otherwise
        Pp = 0.7071*Pt; Pn = Pp; Pd = Pn;
end

```

```

Nc = Ds*(1 - 2*Lc./Ds)/Pp;
Sm = Lbc*(Ds - Dotl + (Pt-D)*(Dotl-D)/Pd);
Nres = D*w*1e3/(Visc*Sm); % Shell-side Reynolds number
Pr = Cp*Visc/Xk; Ptd = Pt/D;
% Heat transfer coefficients for ideal tube bank
% (Nres: Reynolds number of shell-side fluid)
if Nres >= 1e4, J = 1; c1 = 1.25;
elseif Nres >= 1e3, J = 2; c1 = 1.25;
elseif Nres >= 100, J = 3; c1 = 1.25;
elseif Nres >= 10, J = 4; c1 = 1.35;
else J = 5; c1 = 1.35; end
a = a3(Layout)/(1 + 0.14*Nres^a4(Layout));
Hj = a1(Layout,J) * (1.33/Ptd)^a * Nres^a2(Layout,J);
Hsi = Hj*Cp*Phi*w*Pr^(-0.6667) / (Sm*1e-6);
% Correction factor for baffle configuration effects
adm = (Ds - 2*Lc)/Dotl; adm1 = acos(adm);
Ftc = (pi + 2*adm*sin(adm1) - 2*adm1)/pi;
Phic = Ftc + 0.54*(1 - Ftc)^0.345;
% Correction factor for baffle leakage effects
Stb = 0.6223*D*(1 + Ftc)*Nt;
Ssb = Ds*Dsb*0.5*(pi - acos(1 - 2*Lc/Ds));
R1 = (Stb + Ssb)/Sm; R2 = Ssb/(Ssb + Stb);
Phil = 0.44*(1-R2) + (1-0.44*(1-R2))*exp(-2.2*R1);
% Correction factor for bundle bypassing
Fbp = (Ds - Dotl)*Lbc/Sm; Nsc = NSS/Nc;
if Nsc >= 0.5, Phib = 1;
else
if NSS == 0, c2 = 0; else c2 = (2*Nsc)^0.3333; end
Phib = exp(-c1*c2*Fbp);
end
Nb = 1e3*L/Lbc + 1;
% Correction factor for adverse temperature gradient buildup
if Nres >= 100, Phir = 1;
else
Ncw = 0.8*Lc/Pp; Phs = 1.51/((Nc+Ncw)*(Nb+1))^0.18;
if Nres <= 20
    Phir = Phs;
elseif Nres <= 100
    Phir = Phs - (1-Phs)*(0.25-0.0125*Nres);
end
if Phir <= 0.4, Phir = Phs; end
end
% Correction factor due to unequal baffle spacing at inlet and outlet
if Nres >= 100, An = 0.6; else An = 0.333; end
Phis = (Nb-1 + (Lbin/Lbc)^(1-An) + (Lbout/Lbc)^(1-An)) / (Nb-1 + ... + (Lbin+Lbout)/Lbc);
% Shell-side heat transfer coefficient
Hshell = Hsi*Phic*Phil*Phib*Phir*Phis;
end

```

The pressure drops for the tube side and shell side are calculated by the functions *dptube* and *dpshell*, respectively.

dptube.m

```

function DPtube = dptube(D,L,rf,v,rho,Visc,Phi,Npass)
% Calculate tube-side pressure drop
% input:

```

```

% D: tube inside diameter (mm)
% L: tube length (m)
% rf: tube roughness (mm)
% v: tube-side flow rate (m/s)
% rho: fluid density (kg/m^3)
% Visc: fluid viscosity (Ns/m^2)
% Phi: viscosity correction factor
% Npass: number of tube passes in bundle
% output:
% DPtube: tube-side pressure drop (N/m^2)
g = 9.81; rfD = rf/D; Dm = 1e-3*D;
Nre = Dm*v*rho/Visc;
if Nre <= 2100 % Laminar flow
    f = 16/Nre;
elseif Nre <= 4000 % Zigrang & Sylvester friction factor correlation eqn.
    t1 = rfD/3.7; t2 = 5.02/Nre;
    ftm = log10(t1 - t2*log10(t1 + 13/Nre));
    f = 1/(4*log10(t1) + t2*ftm)^2;
else % Round friction factor correlation eqn.
    f = 1/(3.6*log10(Nre/(0.135*(Nre*rfD + 6.5))))^2;
end
pD = 2*f*rho*v^2*L/Dm;
dp1 = pD/Phi; dp2 = 4*rho*L*v^2/(2*g);
DPtube = (dp1 + dp2)*Npass;
end

dpshell.m
function DPs = dpshell(D,Ds,L,Lbc,Lbin,Lbout,Lc,Dotl,Dsb,Pt,Nt,Nss,w,rho,
    Visc,Phi,Layout)
% Calculate shell-side pressure drop
% input:
% D: tube outside diameter (mm)
% Ds: shell inside diameter (mm)
% L: tube length (m)
% Lbc: central baffle spacing (mm)
% Lbin: inlet baffle spacing (mm)
% Lbout: outlet baffle spacing (mm)
% Lc: baffle cut (mm)
% Dotl: shell outer tube limit (mm)
% Dsb: shell-baffle clearance (mm)
% Pt: tube pitch (mm)
% Nt: total number of tubes in the bundle
% Nss: number pairs of sealing strips
% w: shell-side flow rate (kg/s)
% rho: fluid density (kg/m^3)
% Visc: shell-side fluid viscosity (Ns/m^2)
% Phi: viscosity correction factor
% Layout: tube layout (1:triangular, 2:in-line square, 3:rotated square)
% output:
% DPs: shell-side pressure drop
% Correlational coefficients for tube arrangement
b1 = [0.372 0.486 4.570 45.100 48.000; 0.391 0.0815 6.090 32.100 35.000;
    0.303 0.333 3.500 26.200 32.000];
b2 = -[0.123 0.152 0.476 0.973 1.000; 0.148 -0.022 0.602 0.963 1.000;
    0.126 0.136 0.476 0.913 1.000];
b3 = [7.00 6.30 6.59];
b4 = [0.500 0.378 0.520];

```

```

switch Layout
    case 1
        Pp = 0.866*Pt; Pn = Pt/2; Pd = Pt;
    case 2
        Pp = Pt; Pn = Pt; Pd = Pn;
    otherwise
        Pp = 0.7071*Pt; Pn = Pp; Pd = Pn;
end
% Friction factor for ideal tube-bank
Nc = Ds*(1 - 2*Lc./Ds)/Pp; Ptd = Pt/D;
Sm = Lbc*(Ds - Dotl + (Pt-D)*(Dotl-D)/Pd);
Nres = D*w*1e3/(Visc*Sm); % Shell-side Reynolds number
if Nres >= 1e4, J = 1; c1 = 3.7;
elseif Nres >= 1e3, J = 2; c1 = 3.7;
elseif Nres >= 100, J = 3; c1 = 3.7;
elseif Nres >= 10, J = 4; c1 = 4.5;
else J = 5; c1 = 4.5; end
b = b3(Layout)/(1 + 0.14*Nres^b4(Layout));
Fj = b1(Layout,J) * (1.33/Ptd)^b * Nres^b2(Layout,J);
Dpbi = 2*Fj*Nc*(w/(Sm*1e-6))^2 / (rho*Phi);
% Pressure drop for ideal window section
bdm = (Ds - 2*Lc)/Dotl; bd1 = acos(bdm); bd2 = 1 - 2*Lc/Ds;
Ftc = (pi + 2*bdm*sin(bd1) - 2*bd1)/pi;
Sw = (Ds^2/4)*(acos(bd2)-bd2*sqrt(1-bd2^2)) - (Nt/8)*(1-Ftc)*pi*D^2;
Dw = 4*Sw/(1.5708*Nt*(1-Ftc)*D + 2*Ds*bd2);
Ncw = 0.81*Lc/Pp;
Gw = 1e6*w/sqrt(Sm*Sw);
if Nres >= 100
    Dpw = 0.5*Gw^2 * (2+0.6*Ncw) / rho;
else
    Dpw = (2.6e4*Visc*(Ncw/(Pt-D) + Lbc/Dw^2) + Gw)*Gw/rho;
end
% Correction factor for baffle leakage effects
Stb = 0.6223*D*(1 + Ftc)*Nt;
Ssb = Ds*Dsb*0.5*(pi - acos(bd2));
R1 = (Stb + Ssb)/Sm; R2 = Ssb/(Ssb + Stb);
Pv = -0.15*(1+R2) + 0.8;
R1 = exp(-1.33*(1 + R2)*R1^Pv);
% Correction factor for bundle bypassing
Fbp = (Ds - Dotl)*Lbc/Sm; Nsc = Nss/Nc;
if Nsc >= 0.5, Rb = 1;
else
    if Nss == 0, c2 = 0; else c2 = (2*Nsc)^0.3333; end
    Rb = exp(-c1*c2*Fbp);
end
% Correction factor due to unequal baffle spacing at inlet and outlet
Nb = 1e3*L/Lbc + 1;
if Nres >= 100, An = 0.2; else An = 1; end
Rs = (Lbin/Lbc)^(2-An) + (Lbout/Lbc)^(2-An);
% Shell-side pressure drop
Dps = Rb*Dpbi*((Nb-1)*R1 + (1+Ncw/Nc)*Rs) + Nb*R1*Dpw;
end

```

There are six critical variables to be specified: T_1 and T_2 (the inlet and outlet temperature of the hot fluid), t_1 and t_2 (the inlet and outlet temperature of the cold fluid), and W_i and W_o (the inlet and outlet mass flow rate of the fluid). If five variables are known, the 6th variable should be determined

TABLE 8.3
Eight Possible Combinations for Two Unknown Variables

Unknown Variables	Problem Type
All variables known	0
t_1, T_1	1
t_1, T_2	2
t_1, W_o	3
t_2, T_1	4
t_2, T_2	5
t_2, W_o	6
W_o, T_1	7
W_o, T_2	8

Source: Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, 1985, p. 295.

beforehand and specified in the input. If two variables are unknown, then there are eight possible combinations. A flag is specified to denote the type of combination as shown in [Table 8.3](#).

The input data required in the heat exchanger calculations consist of the following:

Heat exchanger geometry: N_T, N_{TP}, N_{ss} .

Tube arrangement: $D_s, l_{bc}, l_{in}, l_{out}, l_c, l_s, \delta_{sb}, \delta_{otb}, L, d_o, d_i, p_T$.

Physical properties: two values of reference temperatures, densities, viscosities, thermal conductivities, and heat capacities at the two reference temperatures for tube and shell fluids (liquid or vapor).

Process data: fouling resistances for the tube and shell side, values for six critical variables (up to two trial values are specified if only four variables are known), and a flag for problem type ([Table 8.3](#)).

The script *hxndat* defines the input data. The main script *hxnst* calls the script *hxndat* to specify input data and calls related functions to perform the calculations.

```
% hxnst.m: Shell-and-tube heat exchanger
clear all;
hxndat2; % input data
% Problem type
if ptype > 0
    if ptype <= 3, Nvar = 1;      % calculate tube-side inlet temp. (Ti1)
    elseif ptype <= 6, Nvar = 2;  % calculate tube-side outlet temp. (Ti2)
    else Nvar = 3; end           % calculate tube-side flow rate (Wi)
end

Rw = 1e-3*(Do - Di)/(2*Xkw); % Xkw: inverse of tube-wall heat
conductivity(1/hw)
A = pi*1e-3*Do*(L - 2e-3*Ls)*Nt;
if ptype == 0
    Cpim = hxcp(cpreft,Trt,(Ti1+Ti2)/2); % tube side
    Cpsm = hxcp(cprefs,Trs,(Ts1+Ts2)/2); % shell side
    Qi = Wi*Cpim*(Ti2 - Ti1);   Qs = Ws*Cpsm*(Ts2 - Ts1);
    Qr = abs(Qi/Qs);
```

```

if (abs(1-Qr) > 0.1)
    display('Tube-side and shell-side heat duty differ by more
    than 10%.');
    display('Tube-side heat duty is used as total duty.');
end
Q = Qi;
end
% Iterations until differences in successive values of key variables
% converge to zero.
varC = 10; iter = 0;
while varC > 1e-3
    if ptype > 0
        % Calculation of unknown variables using energy balances
        Cpim = hxcp(cpreft,Trt,(Til+Ti2)/2); % tube-side
        Cpsm = hxcp(cprefs,Trs,(Ts1+Ts2)/2); % shell-side
        Qs = Ws*Cpsm*(Ts2 - Ts1); Qi = -Qs;
        iter1 = 0;
        switch Nvar
            case 1 % calculates Til(tube-side inlet temperature)
                Tilnew = Ti2 - Qi/(Wi*Cpim); crT = 10;
                while crT >= 1e-3
                    Til = Tilnew;
                    Cpim = hxcp(cpreft,Trt,(Til+Ti2)/2);
                    Tilnew = Ti2 - Qi/(Wi*Cpim);
                    crT = abs((Tilnew - Til)/Tilnew);
                    iter1 = iter1 + 1;
                end
            case 2 % calculates Ti2(tube-side outlet temperature)
                Ti2new = Ti1 + Qi/(Wi*Cpim); crT = 10;
                while crT >= 1e-3
                    Ti2 = Ti2new;
                    Cpim = hxcp(cpreft,Trt,(Til+Ti2)/2);
                    Ti2new = Ti1 + Qi/(Wi*Cpim);
                    crT = abs((Ti2new - Ti2)/Ti2new);
                    iter1 = iter1 + 1;
                end
            case 3 % calculates Wi(tube-side flow rate)
                Wi = abs(Qs/(Cpim*(Ti2 - Til)));
        end
    end
    Tib = (Til + Ti2)/2;

    % Shell-side heat transfer coefficient
    Tsb = (Ts1 + Ts2)/2;
    mus = hxvis(murefs,Trs,Tsb,fsS); % shell-side viscosity
    rhos = hxrho(rhorefs,Trs,Tsb,fsS); % shell-side density
    Cps = hxcp(cprefs,Trs,Tsb); % shell-side heat capacity
    xks = hxthc(xkrefs,Trs,Tsb); % shell-side heat conductivity
    Tw = (Tsb + Tib)/2;
    Twnew = Tw; crT = 10;
    while crT >= 1e-3
        if fsS == 1 % liquid
            Phis = (mus/hxvis(murefs,Trs,Tw,1))^(0.14);
        elseif fsS == 2 % gas
            Phis = (Tsb/Tw)^(0.25);
        end
    end
end

```

```

Hs = htcsheell(Do,Ds,L-2e-3*Ls,Lbc,Lbin,Lbout,Lc,Dotl,Dsb,Pt,....
Nt,Nss,Ws,mus,Cps,xks,Phis,Layout);

% tube-side heat conductivity
mut = hxvis(mureft,Trt,Tib,fsT); % tube-side viscosity
rhot = hxrho(rhoreft,Trt,Tib,fsT); % tube-side density
Cpt = hxcp(cpreft,Trt,Tib); % tube-side heat capacity
xkt = hxthc(xkreft,Trt,Tib); % tube-side heat conductivity
Ui = 4e6*Wi*Npass/(pi*rhot*Di^2*Nt);
Rei = 1e-3*Di*Ui*rhot/mut;
Pri = Cpt*mut/xkt;
if fsT == 1
    Phit = (mut/hxvis(mureft,Trt,Tw,1))^0.14;
elseif fsT == 2
    Phit = (Tw/Tib)^0.25;
end
Ht = htctube(Rei,Pri,Di,L,xkt,Phit);

% Estimate tube wall temperature
Tw = Tib + Hs/(Hs + Ht) * (Tsb - Tib);
crT = abs((Tw - Twnew)/Tw);
Twnew = Tw;
end % end while

% Calculate heat duty from heat transfer equations
U = 1/(Do/(Di*Ht) + 1/Hs + Rw + Rds + Rdt);
Dt1 = Ts1 - Ti2; Dt2 = Ts2 - Ti1;
if (Dt1 <= 0 || Dt2 <= 0) break; end
Delt = (Dt1 - Dt2)/log(Dt1/Dt2);
Ft = 1;
if Npass > 1
    R = (Ts1 - Ts2)/(Ti2 - Ti1);
    P = (Ti2 - Ti1)/(Ts1 - Ti1);
    tm = sqrt(R^2 + 1);
    Ft = tm*log((1-P)/(1-R*P))/((R-1)*log((2-P*(R+1-tm))/(2-P*(R+1+tm))));
end
Deltm = Delt*Ft;
if ptype == 0
    Areq = Qi/(U*Deltm); Da = (A - Areq)/A*100;
    break;
end
Q = U*A*Deltm;

% Calculate assumed unknown variable
Sgn = 1;
if Qs < 0, Sgn = -1; end
switch ptype
    case {1,4,7}
        Ts1new = Ts2 - Sgn*Q/(Ws*Cps);
        varC = abs((Ts1 - Ts1new)/Ts1new);
        Ts1 = Ts1new;
    case {2,5,8}
        Ts2new = Ts1 + Sgn*Q/(Ws*Cps);
        varC = abs((Ts2 - Ts2new)/Ts2new);
        Ts2 = Ts2new;

```

```

case {3,6}
    Wsnew = abs(Q/((Ts2 - Ts1)*Cps));
    varC = abs((Ws - Wsnew)/Wsnew);
    Ws = Wsnew;
end
iter = iter + 1;
end

% Pressure drop
Dps = dpshell(Do,Ds,L-2e-3*Ls,Lbc,Lbin,Lbout,Lc,Dotl,Dsb,Pt,Nt,Nss, ...
    Ws,rhos,mus,Phis,Layout);
Dpt = dptube(Di,L,rf,Ui,rhot,mut,Phit,Npass);

% Print results
fprintf('Overall heat transfer coefficient: U = %g(W/m^2/K)\n', U);
fprintf('Heat transfer coefficient: tube-side = %g(W/m^2/K), shell-side = ...
    %g(W/m^2/K)\n', Ht,Hs);
fprintf('Heat duty: Q = %g(W)\n', Q);
fprintf('Pressure drop: tube-side = %g(Pa), shell-side = %g(Pa)\n',
    Dpt,Dps);
fprintf('Tube-side: Ti1 = %g(K), Ti2 = %g(K), flow rate = %g(kg/s)\n',Ti1,Ti2,Wi);
fprintf('Shell-side: Ts1 = %g(K), Ts2 = %g(K), flow rate = ...
    %g(kg/s)\n',Ts1,Ts2,Ws);

```

Example 8.8: Shell-and-Tube Countercurrent Heat Exchanger¹⁷

Liquid benzene flowing at a rate of 4.8 kg/sec is cooled from 353 K in a four-pass shell-and-tube heat exchanger with cooling water flowing in the tubes. Inlet and outlet temperatures of cooling water are measured to be 298 and 303 K, respectively. The heat exchanger geometry is as follows: $d_o = 25.4$ mm, $d_i = 19.86$ mm, $N_T = 86$ (mounted on equilateral triangular pitch with a pitch ratio of 1.25), tube roughness = 0.025 mm, $L = 2$ m, $l_s = 27$ mm, $D_s = 305$ mm, $d_{oil} = 294$ mm, $\delta_{sb} = 4.45$ mm, $N_{ss} = 0$, $l_{bc} = 450$ mm, $l_{in}, l_{out} = 165$ mm, baffle cut for segmental baffles = 25%. Thermal conductivity of wall material is 45 W/m/K. The fouling resistances for the tube side and shell side are 0.00036 and 0.00018 $\text{m}^2\text{K}/\text{W}$, respectively. The physical properties at reference temperatures are shown in Table 8.4.

Determine the exit temperature of benzene, the heat duty, the flow rate of cooling water, and the pressure drops on the shell and tube sides.

Solution

The script *hxndat* defines the input data as follows:

```

% hxndat.m: Shell-and-tube heat exchanger data
% Two values of reference temperatures and physical properties at the two
% reference temperatures for tube and shell fluids

```

TABLE 8.4
Physical Properties at Reference Temperatures

	Tube Side (Water)	Shell Side (Benzene)
Reference temperature (K)	323	283
Density (kg/m ³)	988.1	999.7
Viscosity (mNs/m ²)	0.6	1.26
Heat conductivity (W/m/K)	0.64	0.603
Heat capacity (J/kg/K)	4183	4195

```

Trt = [323 283]; Trs = [375 289]; % reference temperatures (tube and shell) (K)
rhoreft = [988.1 999.7]; rhorefs = [798 885]; % densities (kg/m^3)
mureft = [0.6 1.26]; murefs = [0.258 0.679]; % viscosities (mNs/m^2)
xkref = [0.64 0.603]; xkrefs = [0.126 0.163]; % thermal conductivities
cpref = [4183 4195]; cprefs = [1980 1675]; % heat capacities (J/kg/K)
% Heat exchanger geometry
Do = 25.4; Di = 19.86; % tube outside diameter (Do,mm) and inside diameter (Di,mm)
Xkw = 45; % heat conductivity of tube wall (W/m/K)
L = 2; % tube length (m)
Ls = 27; % tube sheet thickness (mm)
rf = 0.025; % tube roughness (mm)
Nt = 86; % total number of tubes in tube bundle
Layout = 1; % tube layout (1:triangular, 2:in-line square, 3:rotated square)
Pt = 1.25*Do; % tube pitch (mm)
Nss = 0; % number of pairs of sealing strips
Npass = 4; % number of passes
Rdt = 0.00036; Rds = 0.00018; % fouling resistance (m^2*K/W)
Ds = 305; % shell inside diameter (mm)
Dotl = 294; % shell outside tube limit (mm)
Dsb = 4.45; % shell-baffle clearance (mm)
Lbin = 165; Lbout = 165; % inlet and outlet baffle spacing (mm)
Lbc = 450; % central baffle spacing (mm)
Lc = 0.25*Ds; % baffle cut (mm)
% Specification of key variables
Til = 298; Ti2 = 303; % inlet and outlet temperatures for tube-side (K)
Tsl = 353; % Shell-side (hot stream) inlet temperature (K)
Ws = 4.8; % Shell-side (hot stream) mass flow rate (kg/s)
fsT = 1; fsS = 1; % state of tube-side and shell-side fluids (1:liquid, 2:vapor)
ptype = 8; % problem type
Ts2 = 338; % guess outlet temperature of shell-side fluid
Wi = 5; % guess tube-side flow rate

```

The main script *hxnst* produces the following outputs:

```

>> hxnst

Overall heat transfer coefficient: U = 125.291 (W/m^2/K)
Heat transfer coefficient:
tube-side = 529.371 (W/m^2/K), shell-side = 201.457 (W/m^2/K)
Heat duty: Q = 80243 (W)
Pressure drop: tube-side = 232564 (Pa), shell-side = 26610.9 (Pa)
Tube-side: Til = 298 (K), Ti2 = 303 (K), flow rate = 3.77462 (kg/s)
Shell-side: Tsl = 353 (K), Ts2 = 344.138 (K), flow rate = 4.8 (kg/s)

```

8.3.7 DOUBLE-PIPE HEAT EXCHANGER

A double-pipe finned-tube heat exchanger may be used when the heat duty is moderate (that is, $UA < 100,000$), or when one stream is viscous liquid, or when flow rates are small. Heat transfer coefficients and film coefficients for vapor-liquid streams in the tube side and shell side can be determined using simple correlations.¹⁸

1. Tube-side calculation

For liquid:

$$h_i = 0.023 Re^{0.8} Pr^{0.333} \frac{K}{D_{eq}} \left(\frac{\mu}{\mu_{\omega}} \right)^{0.14} \quad \left(\text{where } \frac{\mu}{\mu_{\omega}} \approx 1 \right), \quad D_{eq} = D_{ti}, \quad Re = \frac{GD_{eq}}{\mu}$$

$$\text{If } Re < 10,000, \quad h_i = J \cdot Pr^{0.333} \frac{K}{D_{eq}}$$

For vapor:

$$h_i = \frac{0.0144 C_p G^{0.8}}{D_{eq}^{0.2}}, \quad \frac{1}{h_{if}} = \frac{1}{h_i} + F_f \quad (\text{fouling correction})$$

2. Shell-side calculation (bare tube): Use $D_{eq} = D_{si} - D_{to}$ in the tube-side calculation.

3. Overall heat transfer coefficient

$$\frac{1}{U_o} = \frac{D_{to} - D_{ti}}{K} + h_{if} \frac{A_i}{A_o} \bigg|_{tube} h_{if} \bigg|_{shell}, \quad \frac{A_i}{A_o} = \frac{D_{ti}}{D_{to}}$$

4. Shell-side calculation (finned tube): In the tube-side calculation, set

$$D_{eq} = \frac{4N_f}{\pi(D_{si} + D_{to}) - N\theta + A_f}, \quad N_f = C_s - \frac{\theta}{2} A_f, \quad C_s = \frac{\pi}{4} (D_{si}^2 - D_{to}^2), \quad A_f = 2HN$$

$$A_o = \pi D_{to} + A_f, \quad X = H \sqrt{\frac{h_{if} \big|_{shell}}{6K\theta}}, \quad \omega = \frac{\tanh X}{X} = \frac{1}{X} \left(\frac{e^X - e^{-X}}{e^X + e^{-X}} \right), \quad \omega' = \omega \left(\frac{A_f}{A_o} \right) + \left(1 - \frac{A_f}{A_o} \right),$$

$$h_{ifd} = h_{if} \omega', \quad \frac{1}{U_o} = \frac{D_{to} - D_{ti}}{K} \bigg|_{tubewall} + \frac{h_{if} A_i}{A_o} \bigg|_{tube} + h_{ifd} \bigg|_{shell}, \quad \frac{A_i}{A_o} = \frac{\pi D_{ti}}{\pi D_{to} + A_f}$$

where

h_i is the heat transfer film coefficient (Btu/(ft² °F hr))

K is the thermal conductivity (Btu/(ft °F hr))

Re is the Reynolds number

Pr is the Prandtl number

D_{eq} is the equivalent diameter (ft)

μ, μ_ω is the fluid viscosity (cP)

J is the heat transfer factor

G is the mass flow rate (lb/(hr ft²))

C_p is the fluid heat capacity (Btu/(lb °F))

A is the cross-sectional area (ft²)

D_{ti} is the tube inside diameter (in.)

F_f is the fouling factor

h_{if} is the heat transfer film coefficient corrected for fouling (Btu/(ft² °F hr))

D_{to} is the tube outside diameter (in.)

D_{si} is the shell inside diameter (in.)

A_i is the inside heat transfer area for the tube (ft²/ft)

A_o is the outside heat transfer area for the tube (ft²/ft)

U_o is the overall heat transfer coefficient (Btu/(ft² °F hr))

N_f is the shell-side net free cross-sectional area (ft²)

N is the number of fins

θ is the fin thickness (normally 0.035 in.)

H is the fin height (in.)

C_s is the cross-sectional area for the shell side without fins (ft²)

A_f is the finned transfer area (ft²)

X is the parameter used in the fin efficiency calculation

ω is the fin efficiency

ω' is the effective surface efficiency for the fins

h_{ifd} is the corrected film heat transfer rate (Btu/(ft² °F hr))

Example 8.9: Finned-Tube Heat Transfer Coefficient

A double-pipe finned-tube heat exchanger is to be used to cool 30°API oil. Determine the tube-side and shell-side heat transfer coefficient and the overall heat transfer coefficient. The operating conditions and data are as follows.

Tube side: flow rate of cooling water = 26,740 lb/hr, $T_{in} = 85^\circ\text{F}$, $T_{out} = 120^\circ\text{F}$,

$$F_f = 0.002, \quad K = 0.366 \frac{\text{Btu}}{\text{ft}^\circ\text{F hr}}, \quad \mu = 0.72 \text{ cP}, \quad C_p = 1 \frac{\text{Btu}}{\text{lb}^\circ\text{F}}$$

Shell side: oil flow rate = 18,000 lb/hr, $T_{in} = 250^\circ\text{F}$, $T_{out} = 150^\circ\text{F}$, $F_f = 0.002$,

$$K = 0.074 \frac{\text{Btu}}{\text{ft}^\circ\text{F hr}}, \quad \mu = 2.45 \text{ cP}, \quad C_p = 0.518 \frac{\text{Btu}}{\text{lb}^\circ\text{F}}$$

Heat exchanger geometry:

Shell: 3 in. (3.068 in. ID, 3.5 in. OD)

Tube: 1.5 in. (1.61 in. ID, 1.9 in. OD)

Number of fins: 24, height 0.5 in., width 0.035 in.

$$K = 25 \frac{\text{Btu}}{\text{ft}^\circ\text{F hr}}$$

Solution

The script *htcoef* calculates the heat transfer coefficients.

```
% htcoef.m : heat transfer coefficients (double-pipe finned-tube heat
exchanger)
clear all;
% Data
Fft = 0.002; Kt = 0.366; mut = 0.72; Cpt = 1; Jt = 9.75; % Tube
Ffs = 0.002; Ks = 0.074; mus = 2.45; Cps = 0.518; % Shell
Dis = 3.068/12; Dos = 3.5/12; % shell size
Dit = 1.61/12; Dot = 1.9/12; % tube size
N = 24; Hf = 0.5/12; w = 0.035/12; % fin size
Gt = 26740; Gs = 18000; % flow rates
% Tube-side :
Deq = Dit;
Pr = Cpt*(2.4191*mut)/Kt; % 1 cP = 2.4191 lb/(ft-hr)
Re = Gt*Deq/(2.4191*mut);
if (Re < 10000)
    hit = Jt* Pr^0.333 * Kt/Deq;
else
    hit = 0.023 * Re^0.8 * Pr^0.333 * Kt / Deq;
end
hift = 1 / (1/hit + Fft);
% Shell-side (bare tube):
Deq = Dis - Dot; Pr = Cps*(2.4191*mus)/Ks; % 1 cP = 2.4191 lb/(ft-hr)
Re = Gs*Deq/(2.4191*mus);
if (Re < 10000)
    his = Jt* Pr^0.333 * Ks/Deq;
else
    his = 0.023 * Re^0.8 * Pr^0.333 * Ks / Deq;
end
```

```

hifs = 1 / (1/his + Ffs);
% Shell-side (finned-tube):
Af = 2*Hf*N; Cs = pi*(Dis^2 - Dot^2)/4; Nf = Cs - w*Af/2;
Deq = 4*Nf/(pi*(Dis+Dot) - N*w + Af);
Ao = pi*Dot + Af; X = Hf*sqrt(hifs/(6*Kt*w));
e = (exp(X) - exp(-X))/(exp(X) + exp(-X))/X;
ep = e*Af/Ao + 1 - Af/Ao; hifd = hifs*ep;
Ar = pi*Dot/(pi*Dot + Af); Uo = 1/((Dot-Dit)/Kt + hift*Ar + hifd);
% Output
fprintf('\nShell-side heat transfer coefficient: %10.7f\n', hifd);
fprintf('Tube-side heat transfer coefficient: %10.7f\n', hift);
fprintf('Fin efficiency: %10.7f\n', ep);
fprintf('Overall heat transfer coefficient: %10.7f\n', Uo);

```

The script *htcoef* generates the following results:

```

>> htcoef
Shell-side heat transfer coefficient: 12.3624311
Tube-side heat transfer coefficient: 41.0441459
Fin efficiency: 0.5070751
Overall heat transfer coefficient: 0.0516646

```

8.4 HEAT TRACING IN PIPELINES

Pipelines conveying viscous fluids are maintained at an elevated temperature by means of heat tracing. Pipelines containing vapors may also be heat traced to prevent components from condensing out. If the cooling resulting from heat loss from pipelines cannot be tolerated, heat tracing the pipelines becomes necessary.¹⁹

Heat transferred through an insulated pipe involves four resistances: the film resistance on the inside wall of the pipe, heat resistance through the pipe wall, heat resistance through the insulation, and air film resistance on the outside of the insulation. These resistances should be considered to determine the surface temperature of insulated pipe, total heat transferred per 100 ft of pipe, flow rate of hot media, and total number of heat tracers required. The equations used to calculate these quantities are as follows²⁰:

$$T_{m,avg} = \frac{T_{mi} + T_{mo}}{2}, \quad T_a = \frac{T_{m,avg} + T_p}{2}, \quad d_i = r_o + t_A, \quad d_o = d_i + 2t_k,$$

$$Q = \frac{2\pi k_i (T_a - T_s)}{\ln(d_o / d_i)}, \quad Q = \frac{h_a \pi d_o (T_s - T_{air})}{12}, \quad T_s = \frac{T_a + xT_{air}}{x + 1},$$

$$x = \ln\left(\frac{d_o}{d_i}\right) \cdot h_a \cdot \frac{d_o}{24k_i}, \quad h_{rc} = \frac{564}{d_o^{0.19} (273 - T_s + T_{air})},$$

$$w_f = 2.814 - 0.0003885714(T_s - T_{air}) - 0.0000012857(T_s - T_{air})^2, \quad h_a = \omega_f h_{rc}$$

$$Q_t = Q \cdot L, \quad W = \frac{Q_t}{C_p (T_{mi} - T_{mo})}, \quad n_t = \frac{Q}{a (T_{m,avg} - T_p)}, \quad n_{tc} = \frac{Q}{b (T_{m,avg} - T_p)}$$

where

- $T_{m,avg}$ is the average temperature of heating medium (°F)
- T_{mi} is the inlet temperature of heating medium (°F)
- T_{mo} is the outlet temperature of heating medium (°F)
- d_i is the inside diameter of insulation (in.)
- d_o is the outside diameter of insulation (in.)
- r_o is the outside diameter of pipe (in.)
- t_A is the allowance for tracer diameter (in.)
- t_k is the insulation thickness (in.)
- T_a is the average temperature of pipe and tracer (°F)
- T_{air} is the air temperature (°F)
- T_s is the outside surface temperature of the insulation (°F)
- k_i is the thermal conductivity of insulation material (Btu/(ft °F hr))
- h_a is the film heat transfer coefficient to air (Btu/(ft² °F hr))
- h_{rc} is the combined convective and radiative heat transfer coefficient (Btu/(ft² °F hr)), w_f is the wind factor
- Q is the total heat loss from pipeline (Btu/hr)
- L is the total pipe length (ft)
- W is the flow rate of heating medium (lb/hr)
- C_p is the specific heat of heating medium (Btu/(lb °F))
- n_t is the number of tracers required without heat transfer cement
- n_{tc} is the number of tracers required with heat transfer cement

Parameters a and b are thermal conductances from the tracer to the pipe without and with heat transfer cement, respectively, and are given in [Table 8.5](#).

The calculation procedure is as follows:

1. Assume h_a and calculate T_s .
2. Calculate the new h_a and update using the wind factor.
3. Calculate the new T_s .
4. Repeat until h_a converges.
5. Calculate Q , W , n_t , n_{tc} .

The script *pipeTracer* executes this procedure.

```
% pipeTracer.m: determine heat loss and tracing requirements
clear all
% Input data
Tmi = input('Inlet temperature of heating medium(deg.F): ') ;
```

TABLE 8.5
Thermal Conductance Parameters

Tube Size (in.)	a	b
3/8	0.295	3.44
1/2	0.393	4.58
5/8	0.490	5.73

```

Tmo = input('outlet temperature of heating medium(deg.F): ');
Cp = input('Specific heat of heating medium(Btu/lb/F): ');
Tp = input('Pipe temperature(deg.F): ');
Tair = input('Air temperature(deg.F): ');
ro = input('Outside diameter of pipe(in): ');
L = input('Pipe length(ft): ');
tt = input('Tracer size(in): ');
tk = input('Insulation thickness(in): ');
Ki = input('Thermal conductivity of insulation(Btu/(hr-ft-F)): ');
% Parameters:
epsh = 1e-4; tA = 1.25;
Tmavg = (Tmi +Tmo)/2; Ta = (Tmavg + Tp)/2;
Di = ro + tA; Do = Di + 2*tk;
if (tt <= 3/8)
    a = 0.295; b = 3.44;
elseif (tt <= 0.5)
    a = 0.393; b = 4.58;
else
    a = 0.49; b = 5.73;
end
% Determine ha (guess initial value)
ha = 4; hai = 0;
while (abs(ha-hai) >= epsh)
    hai = ha; x = (ha*Do/24/Ki)*log(Do/Di);
    Ts = (Ta + x*Tair)/(x+1);
    wf = 2.814-0.0003885714*(Ts-Tair)-0.0000012857*(Ts-Tair)^2;
    ha = 564*wf/(Do^0.19 * (273-Ts+Tair));
end
% Heat loss and tracers
Q = ha*pi*Do*(Ts-Tair)/12; Qt = Q*L;
W = Qt / (Cp*(Tmi-Tmo)); nt = Q / (a*(Tmavg-Tp));
ntc = Q / (b*(Tmavg-Tp));
% Output
fprintf('\nFilm heat transfer coefficient to air = %10.7f Btu/(hr-F-
    ft^2)\n', ha);
fprintf('Total heat loss = %12.5f Btu/hr\n', Qt);
fprintf('Outside surface temperature = %12.5f deg.F\n', Ts);
fprintf('Flow rate of hot medium = %12.5f Btu/hr\n', W);
fprintf('Tracers(without heat-transfer cement) = %3.1f\n', nt);
fprintf('Tracers(with heat-transfer cement) = %3.1f\n', ntc);

```

Example 8.10: Heat Tracer Calculation

Determine the number of tracers required to maintain 100 ft of a 6 in. pipe (outside diameter = 6.065 in.) at 500°F. Hot tracing medium is available at 625°F and has a heat capacity of 0.53 Btu/(lb °F). The pipeline is covered with 2.5 in. of insulation, which has a thermal conductivity of 0.037 Btu/(ft °F hr). The wind speed is 20 mph and the air temperature is 0°F. The hot medium outlet temperature is 550°F. Use 1/2 in. tracers.

Solution

The script *pipeTracer* calculates the number of tracers and total heat loss. The script *pipeTracer* produces the following outputs:

```

>> pipeTracer
Inlet temperature of heating medium(deg.F): 625
outlet temperature of heating medium(deg.F): 550
Specific heat of heating medium(Btu/lb/F): 0.53

```

Pipe temperature(deg.F) : 500
 Air temperature(deg.F) : 0
 Outside diameter of pipe(in) : 6.065
 Pipe length(ft) : 100
 Tracer size(in) : 0.5
 Insulation thickness(in) : 2.5
 Thermal conductivity of insulation(Btu/(hr-ft-F)) : 0.037

Film heat transfer coefficient to air = 3.8641643 Btu/(hr-F-ft²)
 Total heat loss = 23428.95335 Btu/hr
 Outside surface temperature = 18.80591 deg.F
 Flow rate of hot medium = 589.40763 Btu/hr
 Tracers(without heat-transfer cement) = 6.8
 Tracers(with heat-transfer cement) = 0.6

8.5 AIR COOLER

Air coolers may be used for cooling high-pressure streams where the fluid is inside the tubes. In the design of an air cooler, the number of tubes per row as well as the number of tube rows, air outlet temperature, total surface area of tubes, cooler face area, air flow rate, fan horsepower, and tube bundle width should be determined. Various correlation equations are proposed for design calculations²¹:

$$R = 3.167876 + 3.794767 \ln\left(\frac{T_{PI} - T_{AI}}{U}\right), \quad F_V = 720.85418(0.953)^R,$$

$$T'_{AO} = 0.005U\left(\frac{T_{PO} + T_{PI}}{2} - T_{AI}\right) + T_A, \quad L_{MTD} = \frac{(T_{PO} - T_{AI}) - (T_{PI} - T_{AO})}{\ln\left(\frac{T_{PO} - T_{AI}}{T_{PI} - T_{AO}}\right)}$$

$$A = \frac{Q}{U \cdot L_{MTD}}, \quad F_A = \frac{A}{1.2557445R^{1.0030502}}, \quad T_{AO} = \frac{Q}{1.08F_A F_V} + T_{AI},$$

$$F = F_A F_V, \quad \frac{F_A}{B_{HP}} = 7.4212329 + 12.5342466R, \quad \frac{W_t}{F_A} = 36.4 + 9.35R, \quad W = \frac{F_A}{L}$$

where

R is the number of tube rows

T_{PI} is the fluid inlet temperature (°F)

T_{AI} is the air inlet temperature (°F)

A is the bare-tube surface area (ft²)

F_A is the face area of tube bundle (ft²)

F_V is the face velocity of air (ft/min)

T'_{AO} is the estimated air outlet temperature (°F)

U is the overall heat transfer coefficient (Btu/(hr ft² °F))

T_{PO} is the fluid outlet temperature (°F)

L_{MTD} is the log-mean temperature difference (°F)

Q is the heat duty (Btu/hr)

T_{AO} is the calculated air outlet temperature (°F)

F is the air flow rate over tubes (std ft³/min)

B_{HP} is the fan horsepower

W_t is the air cooler weight (lb)

W is the tube bundle width (ft)

L is the tube length (ft)

The calculation procedure is as follows:

1. Input the data: T_{PT} , T_{PO} , T_{AI} , U , Q .
2. Calculate R .
3. Calculate F_V , T'_{AO} and set $T_{AO} = T'_{AO}$.
4. Calculate L_{MTD} , A , F_A , T_{AO} .
5. If $|T_{AO} - T'_{AO}| > \epsilon$, go to 4.
6. Calculate F , $\frac{F_A}{B_{HP}}$, $\frac{W_t}{F_A}$, W .

The script *airCooler* performs this procedure.

```
% airCooler.m: preliminary design of air cooler
clear all;
% Data input
Tpi = input('Inlet fluid temperature(deg.F) : ');
Tpo = input('Outlet fluid temperature(deg.F) : ');
Tai = input('Inlet air temperature(deg.F) : ');
U = input('Overall heat transfer coefficient(Btu/hr/ft^2/deg.F) : ');
Q = input('Heat load(Btu/hr) : ');
L = input('Tube length(ft) : ');
% Parameters:
epst = 1e-4; R = 3.167876 + 3.794767*log((Tpi-Tai)/U);
Fv = 720.85418 * 0.953^R; Tao = 0.005*U*((Tpo + Tpi)/2 - Tai) + Tai;
Taop = Tao + 10;
% Check convergence
while (abs(Tao-Taop) >= epst)
Lmtd = ((Tpo-Tai)-(Tpi-Tao)) / (log((Tpo-Tai)/(Tpi-Tao)));
A = Q/(U*Lmtd); Fa = A/(1.2557445*(R^1.0030502));
Taop = Tao; Tao = Q/(1.08*Fa*Fv) + Tai;
end
% Output variables
F = Fa*Fv; Hp = Fa/(7.4212329+12.5342466*R);
Wt = Fa*(36.4+9.35*R); W = Fa/L;
% Print results
fprintf('\nOutlet air temperature = %10.5f deg.F\n', Tao);
fprintf('Air flow over tube = %12.5f ft^3/min\n', F);
fprintf('Tube bundle width = %9.5f ft\n', W);
fprintf('Log-mean temperature difference = %10.5f deg.F\n', Lmtd);
fprintf('Air cooler weight = %12.5f lb\n', Wt);
fprintf('Bare-tube surface area = %12.5f ft^2\n', A);
fprintf('Fan horsepower = %9.5f\n', Hp);
```

Example 8.11: Air Cooler Calculation

A light hydrocarbon liquid is to be cooled from 160°F to 125°F using an air cooler. The heat transfer rate is 4.55 MM Btu/hr (4,550,000 Btu/hr) and the estimated overall heat transfer is 55 Btu/(hr ft² °F). The dry bulb air temperature (inlet air temperature) is 95°F and the tube length is 40 ft. Perform a preliminary air cooler design.

Solution

The script *airCooler* generates the following results:

```
>> airCooler
Inlet fluid temperature(deg.F) : 160
```

Outlet fluid temperature(deg.F) : 125
 Inlet air temperature(deg.F) : 95
 Overall heat transfer coefficient(Btu/hr/ft²/deg.F) : 55
 Heat load(Btu/hr) : 4550000
 Tube length(ft) : 40

Outlet air temperature = 110.78583 deg.F
 Air flow over tube = 266882.56061 ft³/min
 Tube bundle width = 11.11469 ft
 Log-mean temperature difference = 38.81778 deg.F
 Air cooler weight = 31986.71312 lb
 Bare-tube surface area = 2131.16960 ft²
 Fan horsepower = 8.07255

PROBLEMS

- 8.1** A low-pressure saturated steam flows in an insulated 2 in. schedule 40 steel pipe with a thermal conductivity of 26 Btu/(hr·ft·°F) at 60 psia (292.73°F) as shown in [Figure P8.1](#). The pipe insulation has a thermal conductivity of 0.05 Btu/(hr·ft·°F). The steam side heat transfer coefficient and the air side heat transfer coefficient are $h_i = 2000 \text{ Btu}/(\text{hr} \cdot \text{ft}^2 \cdot \text{°F})$ and $h_o = 4 \text{ Btu}/(\text{hr} \cdot \text{ft}^2 \cdot \text{°F})$, respectively. The air temperature is 70°F.²²
1. Determine the heat flux q to the outside per foot if the insulation is 1 in. thick.
 2. Determine the thickness of the insulation (in.) required to keep the heat flux q at 50 Btu/hr.
- 8.2** A horizontal steel pipe with an external diameter of 0.033 m carries steam at $T_s = 450 \text{ K}$ as shown in [Figure P8.2](#). The pipe is to be insulated with a material whose thermal conductivity k (W/(m·K)) is given by a function of temperature T (K) as²³

$$k = 6.6667 \times 10^{-10} T^3 - 1.54 \times 10^{-6} T^2 + 1.08976 \times 10^{-3} T - 0.14457$$

The external heat transfer coefficient from the surface of a cylinder due to natural convection in air is approximated by²⁴

$$h = 1.32 \left(\frac{|\Delta T|}{D} \right)^{1/4}$$

where

ΔT (K) is the temperature difference between the surface and the air

D (m) is the insulated cylinder diameter

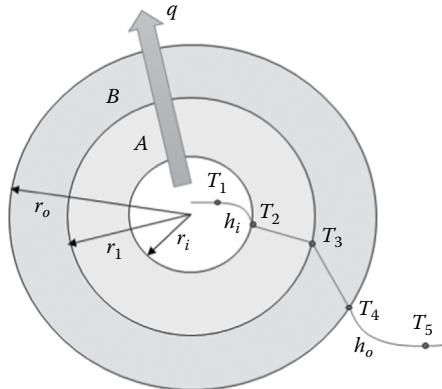


FIGURE P8.1 Heat transfer through multilayer cylinders. (From Cutlip, M.B. and Shacham, M., *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, 2008, p. 344.)

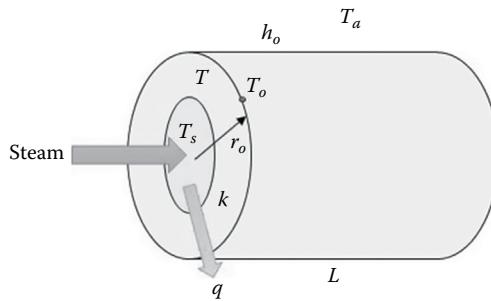


FIGURE P8.2 Heat loss from an insulated pipe. (From Geankoplis, C.J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice-Hall, Boston, MA, 2003, p. 305.)

The ambient temperature is $T_a = 300$ K. The pipe can be assumed to be at the same temperature as the steam due to the high thermal conductivity of steel.

1. Determine the heat loss per meter from the pipe without insulation.

2. Determine the heat loss per meter from the pipe when the insulation thickness is 0.04 m.

- 8.3** The temperature distribution within a two-dimensional metal plate can be expressed by a two-dimensional elliptic partial differential equation of the general form

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f$$

where f is a constant. A thin square metal plate of dimensions $1 \text{ m} \times 1 \text{ m}$ is subject to four heat sources, which maintain the temperature on its four edges as $T(0, y) = 250^\circ\text{C}$, $T(1, y) = 100^\circ\text{C}$, $T(x, 0) = 500^\circ\text{C}$, $T(x, 1) = 25^\circ\text{C}$ (Dirichlet boundary conditions). The flat sides of the plate are insulated so that no heat is transferred through these sides. Determine the temperature profiles within the plate.²⁵

- 8.4** The temperature distribution within a two-dimensional metal plate may be given by a two-dimensional elliptic partial differential equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f$$

where f is a constant. The plate is a thin square metal plate of dimensions $1 \text{ m} \times 1 \text{ m}$. Perfect insulation is installed on two edges (top and right) and the other two edges are maintained at constant temperatures. The set of Dirichlet and Neumann boundary conditions is given by

$$T(0, y) = 250^\circ\text{C}, \quad \left. \frac{\partial T}{\partial x} \right|_{1,y} = 0, \quad T(x, 0) = 500^\circ\text{C}, \quad \left. \frac{\partial T}{\partial x} \right|_{x,1} = 0$$

Determine the temperature profiles within the plate.²⁵

- 8.5** Liquid benzene is to be cooled from 353 to 303 K in a four-pass shell-and-tube heat exchanger, with cooling water flowing in the tubes. The inlet temperature of cooling water is measured to be 298 K and the flow rate of the cooling water is 12 kg/sec. The heat exchanger geometry is as follows: $d_o = 25.4$ mm, $d_i = 19.86$ mm, $N_T = 86$ (mounted on equilateral triangular pitch with a pitch ratio of 1.25), tube roughness = 0.025 mm, $L = 2$ m, $l_s = 27$ mm, $D_s = 305$ mm, $d_{otl} = 294$ mm, $\delta_{sb} = 4.45$ mm, $N_{ss} = 0$, $l_{bc} = 450$ mm, l_{in} ,

TABLE P8.5
Physical Properties at Reference Temperatures

	Tube Side (Water)	Shell Side (Benzene)
Reference temperature (K)	323	283
Density (kg/m ³)	988.1	999.7
Viscosity (mNs/m ²)	0.6	1.26
Heat conductivity (W/m/K)	0.64	0.603
Heat capacity (J/kg/K)	4183	4195

$l_{out} = 165$ mm, baffle cut for segmental baffles = 25%. The thermal conductivity of the wall material is 45 W/m/K. The fouling resistances for the tube side and shell side are 0.00036 and 0.00018 m²K/W, respectively. The physical properties at reference temperatures are shown in Table P8.5.

Determine the exit temperature of the cooling water, the heat duty, the flow rate of benzene, and the pressure drops on the shell and tube sides.

REFERENCES

1. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 209–210, 2008.
2. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice-Hall, Boston, MA, pp. 246–247, 2003.
3. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 334–335, 2008.
4. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 339, 2008.
5. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, pp. 347–349, 2008.
6. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 347, 2008.
7. Constantinides, A. and N. Mostoufi, *Numerical Methods for Chemical Engineers with MATLAB Applications*, Prentice-Hall, Boston, MA, pp. 370–372, 1999.
8. Constantinides, A. and N. Mostoufi, *Numerical Methods for Chemical Engineers with MATLAB Applications*, Prentice-Hall, Boston, MA, pp. 401–402, 1999.
9. Constantinides, A. and N. Mostoufi, *Numerical Methods for Chemical Engineers with MATLAB Applications*, Prentice-Hall, Boston, MA, pp. 382–385, 1999.
10. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, pp. 596–604, 1995.
11. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, pp. 287–290, 1985.
12. Serth, R. W., *Process Heat Transfer*, Academic Press, Elsevier, Burlington, MA, p. 254, 2007.
13. Serth, R. W., *Process Heat Transfer*, Academic Press, Elsevier, Burlington, MA, p. 248, 2007.
14. Serth, R. W., *Process Heat Transfer*, Academic Press, Elsevier, Burlington, MA, p. 249, 2007.
15. Serth, R. W., *Process Heat Transfer*, Academic Press, Elsevier, Burlington, MA, p. 259, 2007.
16. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, p. 295, 1985.
17. Raman, R., *Chemical Process Computations*, Elsevier Applied Science Publishers, Barking, Essex, UK, p. 298, 1985.
18. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, pp. 632–634, 1995.
19. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, p. 645, 1995.

20. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, pp. 649–652, 1995.
21. Coker, A. K., *Chemical Process Design, Analysis and Simulation*, Gulf Publishing Company, Houston, TX, pp. 640–644, 1995.
22. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 344, 2008.
23. Cutlip, M. B. and M. Shacham, *Problem Solving in Chemical and Biochemical Engineering with POLYMAT, Excel, and MATLAB*, 2nd ed., Prentice-Hall, Boston, MA, p. 346, 2008.
24. Geankoplis, C. J., *Transport Processes and Separation Process Principles*, 4th ed., Prentice-Hall, Boston, MA, p. 305, 2003.
25. Constantinides, A. and N. Mostoufi, *Numerical Methods for Chemical Engineers with MATLAB Applications*, Prentice-Hall, Boston, MA, p. 382, 1999.

9 Process Control

The field of process control has been one of the most important contributors to the development of computer simulation, and today, computer simulations are able to handle increasingly more complex problems in this area. Process control problems, if practiced without the aid of a computer, require a thorough grasp of the basic principles involved together with a knowledge of the techniques available for the approximate solution of problems in process dynamics.

MATLAB® has an excellent collection of commands and functions that are useful in solving process control problems. A brief outline of the basic elements in a typical process control system is described in this chapter together with the MATLAB programs. This chapter covers classical techniques but also includes a discussion of state-space modeling and control, a modern control topic lacking in most chemical process control texts. The use of MATLAB programs allows a more general solution that retains all the nonlinear complexities, as well as an optimization of control configurations by rapid trial and error procedures.

The main objective of this chapter is to provide readers with problem-solving techniques using MATLAB programs. The MATLAB programs presented in this chapter are written with user-friendly comments so that the reader can follow each step easily. The topics covered in this chapter include the Laplace transform and transfer function, the state-space model, the dynamics of processes and feedback control loops, the stability of the control system, and frequency response analysis. The MATLAB programs provided for these topics may be used for academic or industrial applications as well as for undergraduate or graduate courses on chemical process control. Researchers and practicing engineers in the field of chemical process engineering can apply these MATLAB programs in the analysis and design of process control systems.

9.1 LAPLACE TRANSFORM AND TRANSFER FUNCTION

9.1.1 LAPLACE TRANSFORM AND INVERSE LAPLACE TRANSFORM

The Laplace transform of a function $f(t)$ is defined as

$$F(s) = \mathcal{L}\{f(t)\} = \int_0^{\infty} f(t) e^{-st} dt$$

The inverse Laplace transform to find $f(t)$ from the given Laplace transform $F(s)$ is expressed as

$$f(t) = \mathcal{L}^{-1}[F(s)]$$

In the partial fraction expansion method used to find $f(t)$, the polynomial in the denominator of $F(s)$ is broken up into simpler partial fractions. $F(s)$ can be represented as

$$F(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0} + R(s)$$

where $n > m$.

Symbolic variables are used in the Laplace transform of a time-domain function using MATLAB. The built-in functions *laplace* and *ilaplace* are used in the Laplace and inverse Laplace transforms, respectively.

Example 9.1: Laplace Transform

Find Laplace transform of $f(t) = 1 + t + t^2 + \sin at - t \cos bt$.

Solution

First create the symbolic variables t, a, and b using the built-in function *syms* and call the built-in function *laplace*. The following commands produce the Laplace transform of f(t).

```
>> syms t a b;
>> f = 1 + t + t^2 + sin(a*t) - t*cos(b*t);
>> Lf = laplace(f)
Lf =
a/(a^2 + s^2) + 1/(b^2 + s^2) - (2*s^2)/(b^2 + s^2)^2 + 1/s + 1/s^2 + 2/s^3
>> pretty(Lf)
2
a      1      2 s      1      1      2
----- + ----- - ----- + - + - + -
2      2      2      2      2      s      2      3
a + s      b + s      (b + s)      s      s
```

The built-in function *pretty* display symbolic expressions using exponents.

Example 9.2: Inverse Laplace Transform

Find the inverse Laplace transform of $F(s) = 2s/(s^2 + 4s + 1)$.

Solution

Create the symbolic variable s and use the built-in function *ilaplace* to get f(t). The following commands produce inverse Laplace transform f(t) of F(s).

```
>> syms s;
>> F = 2*s / (s^2 + 4*s + 1);
>> f = ilaplace(F)
f =
2*exp(-2*t)*(cosh(3^(1/2)*t) - (2*3^(1/2)*sinh(3^(1/2)*t))/3)
>> pretty(f)
/      1/2      2      3      1/2      1/2      \
|      1/2      2      3      sinh(3      t)  |
2 exp(-2 t) | cosh(3      t) - ----- | /
\                  3                  /
```

9.1.2 PARTIAL FRACTION EXPANSION

In MATLAB, the partial fraction expansion is performed by the built-in function *residue*. If the function $F(s)$ can be expanded into the sum of partial fractions and repeated factors, $F(s)$ can be written as

$$\begin{aligned} F(s) &= \frac{N(s)}{D(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0} \\ &= \frac{r_1}{s - p_1} + \dots + \frac{r_k}{s - p_k} + \frac{r_{q1}}{(s - p_q)} + \frac{r_{q2}}{(s - p_q)^2} + \dots + K(s) \end{aligned}$$

For the partial fraction expansion by MATLAB, the built-in function *residue* can be used as

```
[R P K] = residue(N, D)
```

where $R = [r_1, r_2, \dots]$, $p = [P_1, P_2, \dots]$, and K is a row vector containing the coefficients of $K(s)$.

Example 9.3: Partial Fraction Expansion

$F(s) = (s^3 + 5s^2 + 9s + 7)/(s^2 + 3s + 2)$ can be expanded into $F(s) = (-1/(s + 2)) + (2/(s + 1)) + (s + 2)$. Verify this by using MATLAB.

Solution

The built-in function *residue* can be used to verify this.

```
>> N = [1 5 9 7]; D = [1 3 2];
>> [R P K] = residue(N, D)
R =
-1
2
P =
-2
-1
K =
1 2
```

9.1.3 REPRESENTATION OF TRANSFER FUNCTION

In MATLAB, the Laplace transform can be represented by row vectors of coefficients of the numerator and denominator of the Laplace transform $F(s)$ in descending order. Data are input using bracket [] where numbers are separated by a blank space or a comma (,). If the denominator is factorized, we can use the built-in function *conv* to expand the multiplication of these factors.

Once the coefficient vectors for the denominator and the numerator of a transfer function are specified, the MATLAB built-in function *tf* can be used to get the transfer function in the form of Laplace transform. In this case, other MATLAB commands can easily be used by storing the output of *tf* in a variable.

Example 9.4: Representation of Transfer Function

Use MATLAB to display $G(s) = ((2s + 1)/(s^2 + 3s + 2)) = ((2s + 1)/((s + 1)(s + 2)))$.

Solution

The built-in function *conv* can be used to expand the product of two terms. The following commands represent $G(s)$ in expanded form.

```
>> num = [2 1]; den = conv([1 1], [1 2]);
>> G = tf(num, den)
G =
2 s + 1
-----
s^2 + 3 s + 2
Continuous-time transfer function
```

Example 9.5: Use of the Built-In Function *tf*

Find the transfer function G when the numerator is given by $2s + 1$ and the denominator is given by $s^2 + 3s + 2$.

Solution

The following commands produce the transfer function G :

```
>> num = [2 1]; den = [1 3 2];
>> G = tf(num, den)
G =
2 s + 1
-----
s^2 + 3 s + 2
Continuous-time transfer function.
```

9.2 BLOCK DIAGRAM

Block diagrams consists of unidirectional, operational blocks that represent the transfer function of the variables of interest. The components of a chemical process can be represented as blocks. The interconnected blocks show clearly the flows of information and signal between adjacent process components. [Figure 9.1](#) shows a simple feedback control system represented by a block diagram.

In [Figure 9.1](#), each block represents a process element whose characteristics can be expressed in terms of algebraic or differential equations. Identification of mathematical expressions for each block is one of important tasks a process engineer has to perform.

The MATLAB built-in functions *series*, *parallel*, and *feedback* can be used to synthesize block diagrams. The function *series* is used when the blocks are connected in series, the function *parallel* is used when the blocks are connected in parallel, and the function *feedback* is used to synthesize a block diagram when the output is used as a feedback signal.

Example 9.6: Overall Transfer Function

Find the overall transfer function for simple feedback loop shown in [Figure 9.2](#). In [Figure 9.2](#), $G = (2s + 1)/(s^2 + 3s + 2)$, $H = 1/(s + 1)$.

Solution

The built-in function *feedback* produces the desired overall transfer function:

```
>> ng = [2 1]; dg = [1 3 2]; nh = 1; dh = [1 1];
>> [nt, dt] = feedback(ng, dg, nh, dh, -1);
>> Gcl = tf(nt, dt)
Gcl =
2 s^2 + 3 s + 1
-----
s^3 + 4 s^2 + 7 s + 3
Continuous-time transfer function.
```

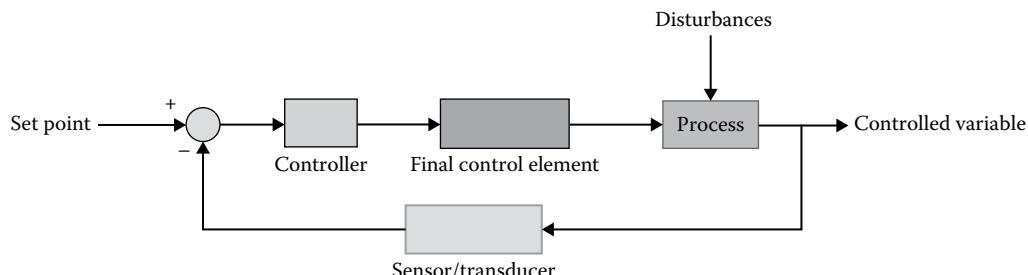


FIGURE 9.1 A block diagram representation of a simple feedback control system.

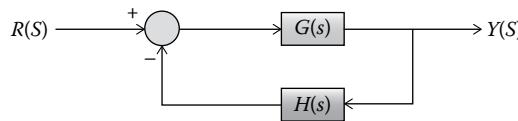


FIGURE 9.2 A simple feedback loop.

The numerator and denominator of G are stored in `ng` and `dg`, respectively, and those of H are stored in `nh` and `dh`, respectively. The numerator and denominator of the overall transfer function are `nt` and `dt`, which are stored in `Gcl`. When the function `feedback` is used, negative or positive feedback should be specified by `-1` (negative feedback) and `+1` (positive feedback).

9.3 STATE-SPACE REPRESENTATION

A set of differential equations can be put in standard vector–matrix form:

$$\dot{\underline{x}}(t) = A\underline{x}(t) + B\underline{u}(t), \quad \underline{y}(t) = C\underline{x}(t) + D\underline{u}(t)$$

where

\underline{x} is the state variable vector

\underline{u} is the input vector

\underline{y} is the output vector

The time derivative is denoted by the overdot. In addition, A is the process matrix, B is the input matrix, C is the output matrix, and D is the feedforward matrix. If the dimension of the vector \underline{x} , \underline{u} , and \underline{y} are n , m , and r , respectively, the dimension of A is $n \times n$, that of B is $n \times m$, that of C is $r \times n$, and that of D is $r \times m$.

The resolvent matrix $\Phi(s)$ is defined as

$$\Phi(s) = (sI - A)^{-1}$$

This gives the time-domain representation $\Phi(t) = e^{At}$, and we have

$$\underline{x}(t) = \Phi(t)\underline{x}(0) + \int_0^t \Phi(t-\tau)B\underline{u}(\tau)d\tau = e^{At}\underline{x}(0) + \int_0^t e^{A(t-\tau)}B\underline{u}(\tau)d\tau$$

The Laplace transformation of the state-space model yields

$$Y(s) = CX(s) = C\Phi(s)B \cdot U(s) = G(s)U(s)$$

where

$$G(s) = C\Phi(s)B = C(sI - A)^{-1}B$$

is the transfer function matrix.

Once the matrices A , B , and C are known, the state-space model can be readily entered in MATLAB by defining the three matrices and using the `ss` command. The command `tf2ss` is used to create a state-space model from a given transfer function model. The command `ss2tf` generates a transfer model from the given state-space model.

Example 9.7: From Transfer Function to State-Space Model

Find a state-space model for the process whose transfer function is given by

$$G = \frac{2s+1}{s^3 + 3s^2 + 2s + 1}.$$

Solution

The command *tf2ss* generates the matrices A, B, C, and D.

```
>> n = [2 1]; d = [1 3 2 1];
>> [A B C D] = tf2ss(n,d)
A =
-3    -2    -1
1     0     0
0     1     0
B =
1
0
0
C =
0     2     1
D =
0
```

Example 9.8: State-Space Representation

Find a state-space representation for the system given by

$$\frac{d^2y}{dt^2} + 1.5 \frac{dy}{dt} + y = \frac{du}{dt} + 2u, \quad y(0) = \frac{dy}{dt}(0) = u(0) = 0$$

Solution

To obtain the matrices A, B, C, and D, we first take the Laplace transform of the differential equation and supply the coefficient vectors of the numerator and denominator in descending order into the command *tf2ss*.

```
>> [A B C D] = tf2ss([1 2], [1 1.5 1])
A =
-1.5000    -1.0000
1.0000        0
B =
1
0
C =
1     2
D =
0
```

9.4 PROCESS DYNAMICS

9.4.1 DYNAMICS OF A 1ST-ORDER PROCESS

A 1st-order process is represented by the 1st-order differential equation

$$\tau \frac{dy}{dt} + y = Kx(t)$$

where

y is the output

x is the input

τ is the time constant of the process

K is the gain of the process

Assume that the initial condition is given by $y(0) = 0$. The Laplace transform of this model yields

$$\frac{Y(s)}{X(s)} = G(s) = \frac{K}{\tau s + 1}$$

The output $Y(s)$ depends on the input $X(s)$. For a step input of magnitude A , the time-domain response is given by

$$Y(t) = KA \left(1 - e^{-t/\tau}\right)$$

For a block pulse input of magnitude H and duration of T , the output $Y(s)$ is given by

$$Y(s) = \frac{KH}{s(\tau s + 1)} \left(1 - e^{-Ts}\right)$$

and the time-domain response can be expressed as

$$Y(t) = KH \left[1 - e^{-t/\tau} - \left\{ 1 - e^{-(t-T)/\tau} \right\} u(t-T) \right]$$

The Laplace transform of a ramp input with a slope of A is $X(s) = A/s^2$. Thus, the output is given by

$$Y(s) = G(s)X(s) = \frac{KA}{s^2(\tau s + 1)} = \frac{KA\tau^2}{\tau s + 1} - \frac{KA\tau}{s} + \frac{KA}{s^2}$$

and the time-domain response is given by

$$Y(t) = KA \left(t + \tau e^{-t/\tau} - \tau \right)$$

A sinusoidal input with an amplitude of A and frequency of ω can be represented as $X(t) = A \sin(\omega t)u(t)$. The Laplace transform of this function is $X(s) = A\omega/(s^2 + \omega^2)$ and the output is given by

$$Y(s) = G(s)X(s) = \frac{KA\omega}{(\tau s + 1)(s^2 + \omega^2)} = \frac{KA}{1 + \tau^2 \omega^2} \left(\frac{\tau^2 \omega}{\tau s + 1} - \frac{\tau \omega s}{s^2 + \omega^2} + \frac{\omega}{s^2 + \omega^2} \right)$$

We can see that the time-domain response can be represented as

$$Y(t) = \frac{KA\omega\tau}{1 + \tau^2 \omega^2} e^{-t/\tau} + \frac{KA}{\sqrt{1 + \tau^2 \omega^2}} \sin(\omega t + \phi)$$

where ϕ is the phase angle given by $\phi = \tan^{-1}(-\tau\omega)$. As $t \rightarrow \infty$, the exponential term goes to zero and the time-domain output $Y(t)$ becomes a sine curve with constant amplitude and frequency:

$$Y(\infty) = \frac{KA}{\sqrt{1+\tau^2\omega^2}} \sin(\omega t + \phi)$$

In this oscillation, the period is given by $T = (2\pi)/\omega$. The frequency of the sinusoidal oscillation is represented either by Hertz (number of cycles per second), Hz, or by radian per unit time, ω .

The step response of a 1st-order process can be obtained by the built-in function *step* in MATLAB. The *step* function plots the unit step response for each input–output pair of the system, assuming that the initial conditions are zero. The *step* function requires the transfer function and response time as input arguments.

The built-in function *impulse* calculates the impulse response of a 1st-order process. The *impulse* function plots the unit impulse response for each input–output pair of the system, assuming that the initial conditions are zero. This function also needs the transfer function and response time as input arguments.

The built-in function *lsim* calculates the response for a linear input. The *lsim* function plots the response of the system to an arbitrary input. This function needs the transfer function, the input form, and the response time as input arguments. The *lsim* function can also be used to calculate the response for a sinusoidal input.

Example 9.9: Step Response of a 1st-Order Process

Plot the unit step response of a 1st-order process during the time interval of $[0, 10]$. The transfer function of the process is given by $3/(2s + 1)$.

Solution

The script *step1* generates the desired step response curve.

```
% step1.m
num = 3; den = [2 1]; t = [0:0.1:10];
y = step(num, den, t);
plot(t, y), grid, title('Step response of a 1st-order process')
xlabel('t(sec)'), ylabel('Response y(t)')
```

Figure 9.3 shows the step response curve of the given 1st-order process.

The step response can also be obtained by the Simulink®. To start the simulation, do the following steps:

1. Click on the “New” icon (>New) in the MATLAB menu bar and select “Simulink Model.”
 2. A new model window appears. In the menu bar of the new editor window, select the “Library Browser” icon ().
 3. Select and place in the new window the “Step” block from the “Sources” library located in the “Simulink” submenu. This can be done by selecting the “Step” block while pressing the left button of the mouse and dragging it to the new window.
 4. Select and place in the new window the “Transfer Fcn” (transfer function) block from the “Continuous” library and the “Scope” block from the “Sinks” library.
 5. Construct a Simulink block diagram for the 1st-order process. In order to connect the blocks in the new editor window, click on a block, locate the mouse on the block to be connected, and click on the left button of the mouse while pressing the “Ctrl” key.
- Figure 9.4 shows the block diagram constructed by connecting the blocks.

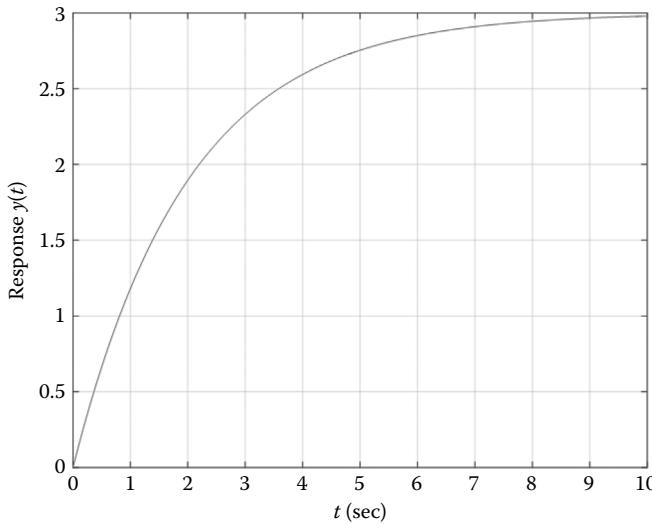


FIGURE 9.3 Step response of a 1st-order process.

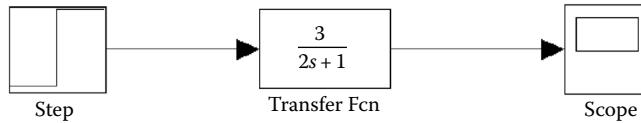


FIGURE 9.4 Simulink® block diagram of a 1st-order process.

6. Specification of the numerator and the denominator of the transfer function can be done in the editor window, which appears by double-clicking on the "Transfer Fcn" block.
7. In the editor window obtained by double-clicking on the "Step" block, set "Step time" to 0 and select the "Apply" button and then press the "OK" button.
8. In the "Simulation" menu of the model editor window, select "Model Configuration Parameters," set the calculation time to 10(sec), and select "Run" in the "Simulation" menu.
9. The response curve can be seen by double-clicking the "Scope" block. [Figure 9.5](#) shows the response curve displayed in the "Scope" block.

Example 9.10: Sinusoidal Response of a 1st-Order Process

Plot the response curve of the 1st-order system $3/(2s + 1)$ to the sinusoidal input $u = \sin(3t)$ during the time interval of $[0, 10]$.

Solution

The script [sin1st.m](#) generates the response curve as shown in [Figure 9.6](#).

```
% sin1st.m: sinusoidal response of the 1st-order process
num = 3; den = [2 1]; G = tf(num,den);
t = [0:0.1:10]; u = sin(3*t); z = t*0; y = lsim(G,u,t);
plot(t,y,t,u,:'), legend('Output y(t)', 'Input u(t)'), hold on
plot(t,z), hold off
xlabel('t(sec)'), ylabel('Response y(t)')
title('Sinusoidal response of 1st-order process')
```

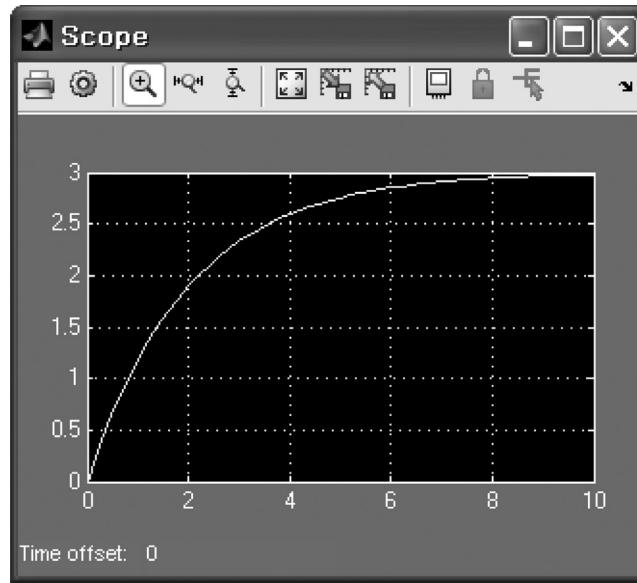


FIGURE 9.5 Step response curve in the Simulation Scope.

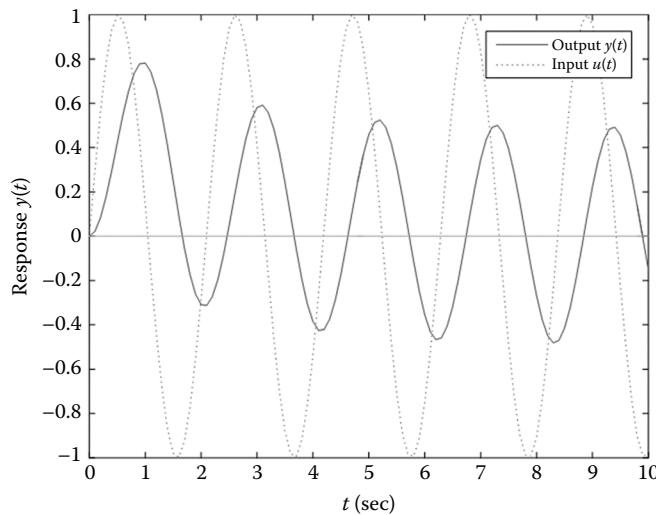


FIGURE 9.6 Sinusoidal response of a 1st-order process.

9.4.2 DYNAMICS OF 2ND-ORDER PROCESSES

A 2nd-order model is represented by a 2nd-order differential equation. The corresponding Laplace transform yields a transfer function with the denominator being a 2nd-order polynomial of s . The response of a 2nd-order process to various inputs can be found by similar methods used in 1st-order processes.

The general form of the transfer function of a 2nd-order process can be expressed as

$$G(s) = \frac{Y(s)}{X(s)} = \frac{K}{\tau^2 s^2 + 2\tau\zeta s + 1}$$

The response $Y(s)$ to the unit step input $X(s) = 1/s$ is given by

$$Y(s) = \frac{K}{s(\tau^2 s^2 + 2\tau\zeta s + 1)} = \frac{K}{\tau^2 s(s - r_1)(s - r_2)} = \frac{K}{\tau^2} \left\{ \frac{A}{s} + \frac{B}{s - r_1} + \frac{C}{s - r_2} \right\}$$

where

$$r_1 = \frac{-\zeta + \sqrt{\zeta^2 - 1}}{\tau}, \quad r_2 = \frac{-\zeta - \sqrt{\zeta^2 - 1}}{\tau}, \quad A = \frac{1}{r_1 r_2}, \quad B = \frac{1}{r_1(r_1 - r_2)}, \quad C = \frac{1}{r_2(r_2 - r_1)}.$$

The characteristics of the roots r_1 and r_2 depend on the decay ratio ζ :

1. $\zeta < 1$: The roots r_1 and r_2 are two complex conjugate poles. This situation is considered underdamped. The time-domain response $Y(t)$ is given by

$$Y(t) = K \left\{ 1 - \frac{1}{\sqrt{1-\zeta^2}} e^{-\zeta t/\tau} \sin \left(\frac{\sqrt{1-\zeta^2}}{\tau} t + \phi \right) \right\}, \quad \phi = \tan^{-1} \left(\frac{\sqrt{1-\zeta^2}}{\zeta} \right)$$

2. $\zeta = 1$: The roots $r_1 = r_2$ are two repeating poles and the time-domain response $Y(t)$ is given by

$$Y(t) = K \left\{ 1 - \left(1 + \frac{t}{\tau} \right) e^{-t/\tau} \right\}$$

3. $\zeta > 1$: In this case, the roots r_1 and r_2 are two distinct real poles, and the time-domain response $Y(t)$ is given by

$$Y(t) = K \left[1 - \frac{1}{2} e^{-\zeta t/\tau} \left\{ \left(1 + \frac{\zeta}{\sqrt{\zeta^2 - 1}} \right) e^{\frac{\sqrt{\zeta^2 - 1}}{\tau} t} + \left(1 - \frac{\zeta}{\sqrt{\zeta^2 - 1}} \right) e^{-\frac{\sqrt{\zeta^2 - 1}}{\tau} t} \right\} \right]$$

For a sinusoidal input $X(t) = A \sin \omega t$, with $X(s) = (A\omega)/(s^2 + \omega^2)$, the output $Y(s)$ is given by $Y(s) = (KA\omega)/((s^2 + \omega^2)(\tau^2 s^2 + 2\tau\zeta s + 1))$. The time-domain response $Y(t)$ can be expressed as

$$Y(t) = e^{-\zeta t/\tau} \left(C_1 \cos \frac{\sqrt{1-\zeta^2}}{\tau} t + C_2 \sin \frac{\sqrt{1-\zeta^2}}{\tau} t \right) + \frac{KA}{\sqrt{(1-\tau^2\omega^2)^2 + (2\tau\zeta\omega)^2}} \sin(\omega t + \phi)$$

where

C_1 and C_2 are constants

$$\phi = -\tan^{-1}((2\tau\zeta\omega)/(1 - \tau^2\omega^2))$$

As $t \rightarrow \infty$, the exponential term goes to zero and the output $Y(t)$ becomes a sinusoidal curve with constant frequency:

$$\lim_{t \rightarrow \infty} Y(t) = \frac{KA}{\sqrt{(1-\tau^2\omega^2)^2 + (2\tau\zeta\omega)^2}} \sin(\omega t + \phi)$$

The step response of a 2nd-order process can be obtained by using the function `step` as in the 1st-order process. The response of a 2nd-order process to impulse, ramp, or other inputs can be found by similar methods used in the 1st-order process.

Example 9.11: Step Response of a 2nd-Order Process

In the 2nd-order process represented by $1/(\tau^2s^2 + 2\tau\zeta s + 1)$, the value of the time constant is $\tau = 0.5$. Plot the step response curves when ζ is 0.5, 1.0, and 1.5 during the time interval [0, 10].

Solution

The script `step2ndpro` generates the desired curves as shown in [Figure 9.7](#).

```
% step2ndpro.m
tau = 0.5; z1 = 0.5; z2 = 1; z3 = 1.5; num = 1; t = [0:0.1:10];
den1 = [tau*tau 2*tau*z1 1]; den2 = [tau*tau 2*tau*z2 1];
den3 = [tau*tau 2*tau*z3 1];
y1 = step(num, den1, t); y2 = step(num, den2, t); y3 = step(num, den3, t);
plot(t, y1, ':', t, y2, '--', t, y3), grid, xlabel('Time t(sec)'),
ylabel('Output y(t)')
title('Step responses of a 2nd-order process');
legend('Underdamped', 'Critically damped', 'Overdamped');
```

The Simulink can be used to obtain the step response of a 2nd-order process. Consider the case when $\zeta = 0.3$. Since $\tau = 0.5$, the transfer function is given by $G(s) = 1/(0.25s^2 + 0.3s + 1)$. Connect blocks in the new model window created by the MATLAB Simulink as shown in [Figure 9.8](#). Double-click on the “Transfer Fcn” block to set the transfer function, and double-click on the “Step” block to set the “Step time” to zero. In the “Simulation” of the editor window, select “Model Configuration Parameters” and set the calculation time to 10(sec). Select “Run” in the “Simulation” menu or click the “Run” icon () in the menu bar to get the step response curve. The result shown in [Figure 9.9](#) can be found by double-clicking the “Scope” block.

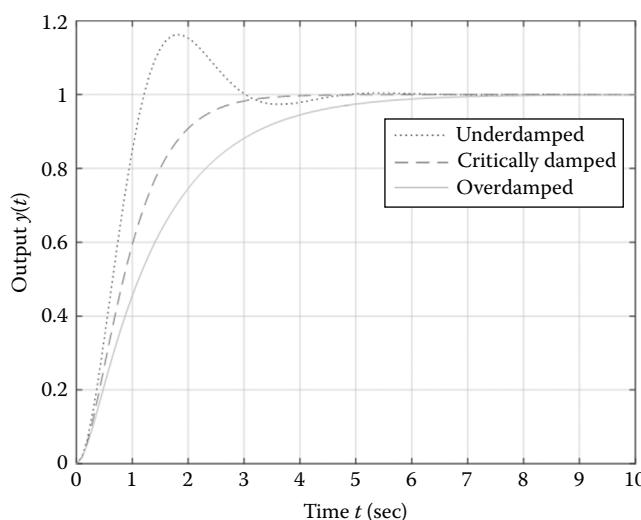


FIGURE 9.7 Step responses of a 2nd-order process.

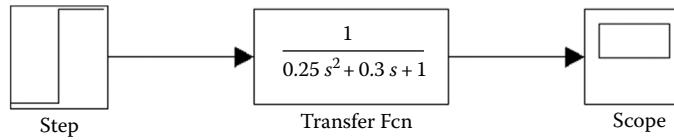


FIGURE 9.8 Simulink® block diagram of a 2nd-order process.

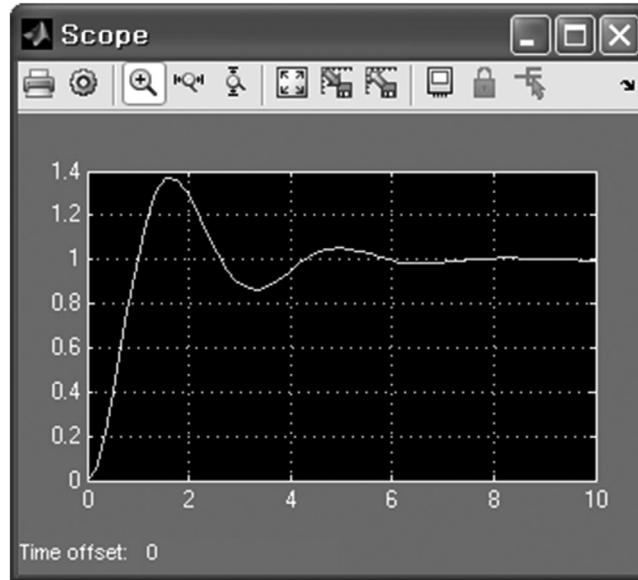


FIGURE 9.9 Step response displayed in the Scope window.

9.4.3 DYNAMICS OF COMPLEX PROCESSES

9.4.3.1 Higher-Order Processes

Higher-order or complex processes are the processes with interconnected 1st-order or 2nd-order processes, with time delays, or with complicated behavior such as inverse responses. The general form of a higher-order process can be written as

$$G(s) = \frac{Y(s)}{X(s)} = \frac{K \prod_{j=1}^m (\tau_{d_j} s + 1)}{\prod_{i=1}^n (\tau_{g_i} s + 1)} \quad (n > m)$$

The unit step ($(X(s) = 1/s)$) response of a higher-order process can be represented as

$$Y(t) = K \left\{ 1 - \sum_{i=1}^n \frac{\prod_{j=1}^m (\tau_{g_i} - \tau_{d_j}) \tau_{g_i}^{n-m-1}}{\prod_{j=1(j \neq i)}^m (\tau_{g_i} - \tau_{g_j})} e^{-t/\tau_{g_i}} \right\}$$

As for the case of 1st- and 2nd-order processes, the step function can be used to get the step response of a higher-order process.

9.4.3.2 Lead/Lag

When $m = n = 1$, the transfer function of a higher-order process becomes

$$G(s) = \frac{Y(s)}{X(s)} = \frac{\tau_d s + 1}{\tau_g s + 1}$$

The term $1/(\tau_g s + 1)$ is called 1st-order lag and the term $\tau_d s + 1$ is called 1st-order lead. The transfer function $G(s)$ is sometimes called lead/lag. The time-domain unit step response of a lead/lag is given by

$$Y(t) = 1 + \left(\frac{\tau_d}{\tau_g} - 1 \right) e^{-t/\tau_g}$$

The built-in function *step* can be used to get the step response of a lead/lag.

9.4.3.3 Time Delay

Most chemical processes involve the movement of mass or energy, and there is a time delay (or dead time) associated with the movement. The transfer function of a time delay of magnitude θ is $e^{-\theta s}$. Thus, the transfer function of a 1st-order process with time delay is given by

$$Y(s) = \frac{K e^{-\theta s}}{\tau s + 1}$$

and the transfer function of a 2nd-order process with time delay is given by

$$Y(s) = \frac{K e^{-\theta s}}{(\tau_1 s + 1)(\tau_2 s + 1)}$$

In MATLAB, the time delay can be represented by using the 'iodelay' option. The basic syntax of this option is

```
delay = 1.6; set(G1, 'iodelay', delay);
```

Example 9.12: Step Response of a Higher-Order Process

Plot the unit step response curves for the higher-order process represented by $1/(2s + 1)^2$, $1/(2s + 1)^4$, $1/(2s + 1)^5$ during the time interval $[0, 20]$.

Solution

The script *highresp* produces the step response curves of a higher-order process.

```
% highresp.m
num = 1; den1 = conv([2 1], [2 1]); den2 = conv(den1, den1);
den3 = conv(den2, [2 1]); t = [0:0.1:20];
y1 = step(num, den1, t); y2 = step(num, den2, t);
y3 = step(num, den3, t);
plot(t, y1, ':', t, y2, '--', t, y3), grid
xlabel('Time t(sec)'), ylabel('Response y(t)')
```

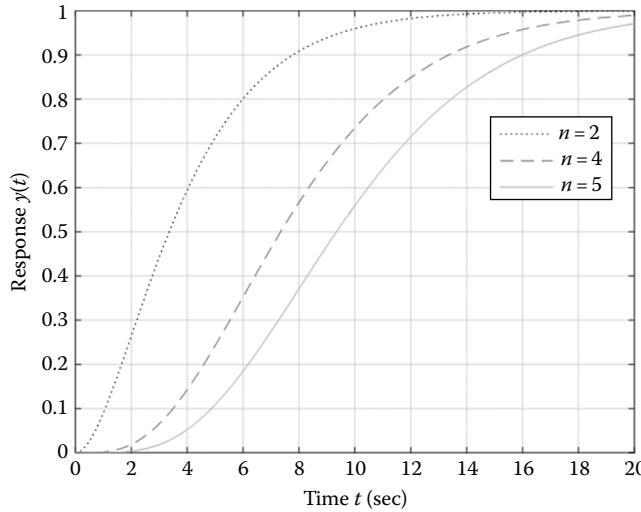


FIGURE 9.10 Step response curves of a higher-order process.

```
title('Step response of higher-order process (n=2, 4, 5)')
legend('n = 2', 'n = 4', 'n = 5', 'location','best')
```

Figure 9.10 shows the step response curves of the higher-order process.

Example 9.13: Step Response of a 1st-Order Plus Time-Delay Process

Plot the unit step response curve of a 1st-order process with time delay, the transfer function of which is given by

$$G(s) = \frac{Y(s)}{X(s)} = \frac{3e^{-1.6s}}{3s+1}$$

Compare the result with the step response of the 4th-order process given by

$$G_2(s) = \frac{Y(s)}{X(s)} = \frac{3}{(0.1s+1)(0.5s+1)(s+1)(3s+1)}$$

Solution

The script *tdelay* uses the 'iodelay' option to generate the required plots shown in Figure 9.11.

```
% tdelay.m
G1 = tf(3, [3 1]); delay = 1.6;
set(G1, 'iodelay', delay);
G2 = tf(3, conv(conv([0.1 1], [0.5 1]), conv([1 1], [3 1])));
step(G1); hold on
step(G2, ':' ); legend('G1', 'G2'), grid, hold off
```

The step response of a higher-order process $G_2(s)$ without time delay is very similar to that of a 1st-order process with time delay. This means that higher-order processes can be represented by a 1st-order process with time delay. In fact, many chemical processes can be effectively represented by the 1st-order plus time-delay model, which can be used in the computer control operation.

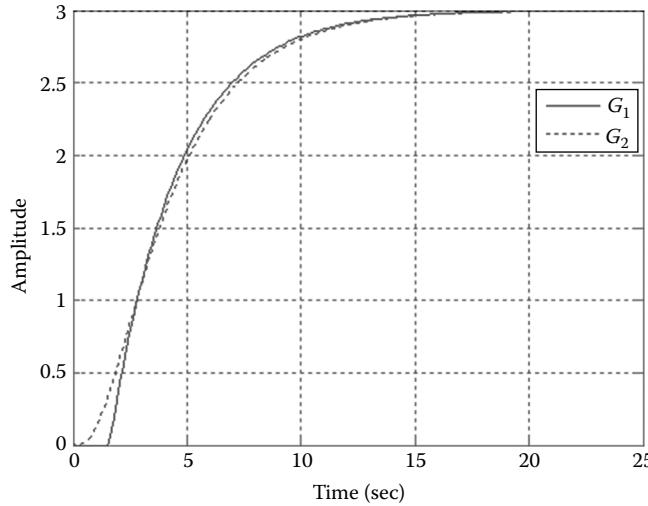


FIGURE 9.11 Step response of a process with time delay.

9.5 DYNAMICS OF FEEDBACK CONTROL LOOPS

9.5.1 CLOSED-LOOP TRANSFER FUNCTION

Figure 9.12 shows a block diagram of a simple closed-loop feedback control system. Each variable is the Laplace transform of a deviation variable, and each block contains the transfer function.

The overall transfer function can be written as

$$C = \frac{G_R G_c G_v G_T G_p}{1 + G_c G_v G_T G_p G_m} R + \frac{G_L}{1 + G_c G_v G_T G_p G_m} L$$

9.5.1.1 Servo Problem

The servo problem refers to the closed-loop system behavior for set-point changes only. In this case, no disturbance change occurs and $L = 0$. From Figure 9.12, it follows that

$$\frac{C}{R} = \frac{G_R G_c G_v G_T G_p}{1 + G_{OL}}, \quad G_{OL} = G_c G_v G_T G_p G_m$$

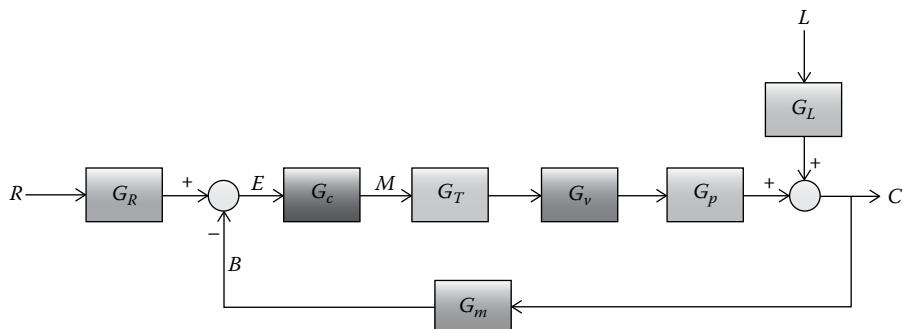


FIGURE 9.12 Block diagram of a closed-loop feedback control system.

9.5.1.2 Regulator Problem

The regulator problem refers to the closed-loop system behavior for disturbance changes only. In this case, no set-point change occurs. Since $R = 0$, the overall closed-loop transfer function can be rearranged to give the closed-loop transfer function for disturbance changes only. From [Figure 9.12](#), it follows that

$$\frac{C}{L} = \frac{G_L}{1 + G_{OL}}$$

The step response of a closed-loop feedback control system can easily be obtained by using the built-in function *step* especially when the process is 1st or 2nd-order and a proportional controller is used.

Example 9.14: Step Response of a Feedback Control System

The overall closed-loop transfer function of a feedback control system is given by

$$C = \frac{K_c}{5s + 1 + K_c} R$$

Calculate and plot the closed-loop response to a unit step change in the set-point for three values of the proportional controller gain: $K_c = 5, 20$, and 50 .

Solution

The script *levelcon* calculates and plots the closed-loop response curves.

```
% levelcon.m
Kc1 = 5; Kc2 = 20; Kc3 = 50; t = [0:0.1:10];
num1 = Kc1; num2 = Kc2; num3 = Kc3;
den1 = [5 1+Kc1]; den2 = [5 1+Kc2]; den3 = [5 1+Kc3];
y1 = step(num1, den1, t); y2 = step(num2, den2, t); y3 = step(num3, den3, t);
plot(t, y1, ':-', t, y2, '--', t, y3), xlabel('Time t(sec)'), ylabel('output y(t)')
legend('Kc = 5', 'Kc = 20', 'Kc = 50')
```

[Figure 9.13](#) shows the step response curves. We can see that the offset decreases as the controller gain increases.

```
>> levelcon
```

Example 9.15: Step Responses for Proportional Control of a 2nd-Order Process

A proportional controller is to be used to control a 2nd-order process given by $G = 0.5/(s(0.5s + 1))$. Calculate and plot the closed-loop response to a unit step change in the set-point for three values of the proportional controller gain: $K_c = 0.5, 1$, and 2 . Assume that $K_v = K_m = 1$.

Solution

The closed-loop transfer function is given by

$$\frac{C}{R} = \frac{0.5K_c}{0.5s^2 + s + 0.5K_c}$$

The script *secpro* asks to input the proportional controller gain and plots the closed-loop response curves.

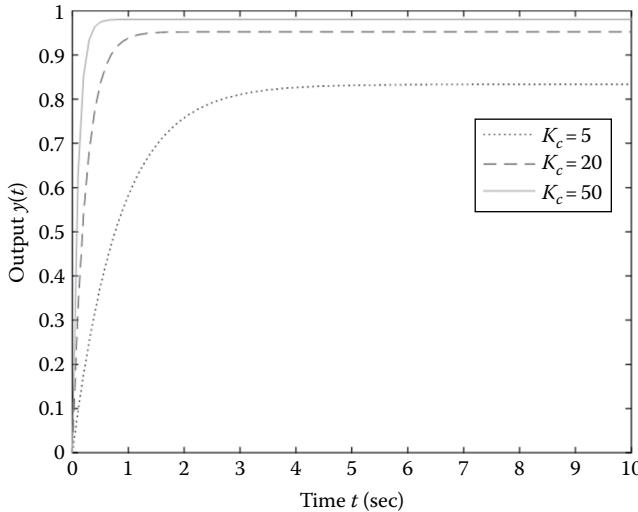


FIGURE 9.13 Step responses for a feedback control system.

```
% secpro.m : feedback control of 2nd-order process using P-controller
% Input controller gain
Kc = input('Controller gain = ');
num = 0.5*Kc; den = [0.5 1 0.5*Kc];
% Unit step response
t = [0:0.1:10]; y = step(num, den, t);
plot(t, y), grid, xlabel('Time t(sec)'), ylabel('Output C(t)');
```

The script *usesecpro* calls the script *secpro* to input the controller gains and plot the step response curves.

```
% usesecpro.m: calls secpro.m to plot step response curve
secpro
hold on, secpro
secpro, hold off
text(2.1, 1.1, 'Kc=0.5'), text(4.1, 0.86, 'Kc=1'), text(5.1, 0.68, 'Kc=2')
```

The script *usesecpro* generates the step response curves as shown in [Figure 9.14](#).

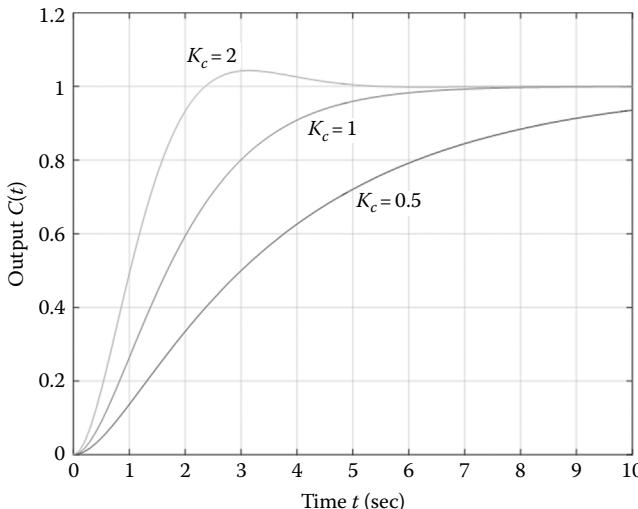


FIGURE 9.14 Step responses for a feedback control of 2nd-order process.

```
>> usesecpro
Controller gain = 0.5
Controller gain = 1
Controller gain = 2
```

Figure 9.14 indicates that increasing K_c tends to speed up the response. But, the closed-loop response becomes more oscillatory as K_c is increased. This means that the decay ratio becomes less than 1 when K_c is greater than a certain value.

Example 9.16: Step Response of a Feedback Control System Using a PI Controller

A proportional–integral (PI) controller is used in a feedback control system. The process transfer function is given by $G_p(s) = 5/(s + 1)(2s + 1)$, the gain of the control valve is $K_v = 0.01$, and the gain of the sensor/transducer is $K_m = 20$. Use the Simulink to find the step response curve to a unit step change in the set-point. The transfer function of the PI controller is given by $G_c(s) = 2(1 + (1/5s))$.

Solution

The Simulink block diagram is shown in Figure 9.15. The scope output is shown in Figure 9.16.

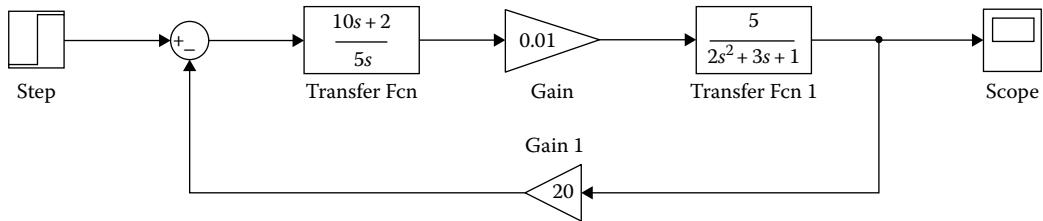


FIGURE 9.15 The Simulink® block diagram for a feedback system using a PI controller.

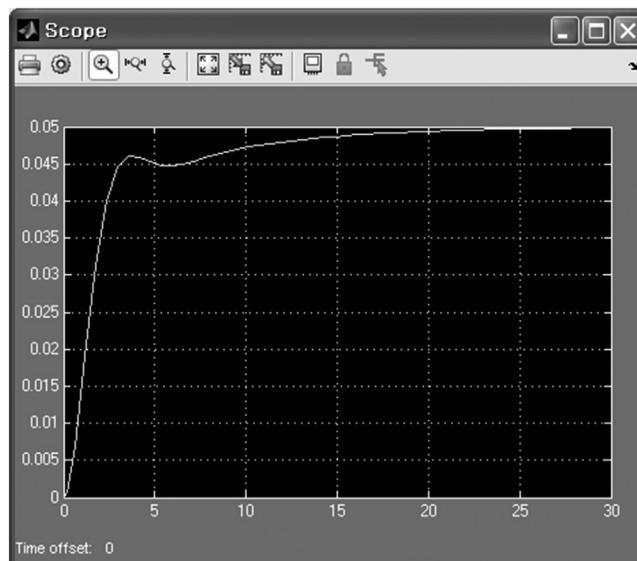


FIGURE 9.16 The scope output for a feedback system using a PI controller.

Example 9.17: Step Response of a Feedback Control System Using a PID Controller

Figure 9.17 shows a simple feedback control system where the process transfer function is $G_p(s) = 3/(4s^2 + s + 1)$ and a proportional–integral–derivative (PID) controller is used. The controller transfer function is $G_c(s) = K_c(1 + (1/\tau_I s) + \tau_D s)$. Calculate and plot the closed-loop response to a unit step change in the set-point for various values of τ_I and τ_D while the value of K_c being kept constant.

Solution

The script *tunpid* generates step response curves for different values of τ_D and τ_I , respectively, as shown in Figure 9.18.

```
% tunpid.m
tau = 2; zeta = 0.25; Kp = 3; Kc = 5;
h(1,:) = '-'; h(2,:) = ':'; h(3,:) = '-.'; h(4,:) = '--';
t0 = 0; delt = 0.1; fint = 20; ms = 1;

% Constant reset time (tauI)
tauI = 1;
tauD = [0.5 1 5 10]; % try 4 different tauD
subplot(1,2,1)
for i = 1:length(tauD)
    num = Kc*Kp*[tauI*tauD(i) tauI 1]; d1 = tauI*tau^2;
    d2 = 2*tauI*tau*zeta+Kc*Kp*tauI*tauD(i);
    d3 = tauI*(1+Kc*Kp); d4 = Kc*Kp; den = [d1 d2 d3 d4];
    [y,t] = stepnp(num, den, t0, delt, fint, ms);
    plot(t,y,h(i,:)), hold on
end
```

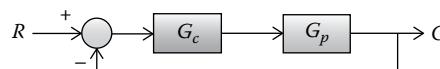


FIGURE 9.17 Simple feedback system.

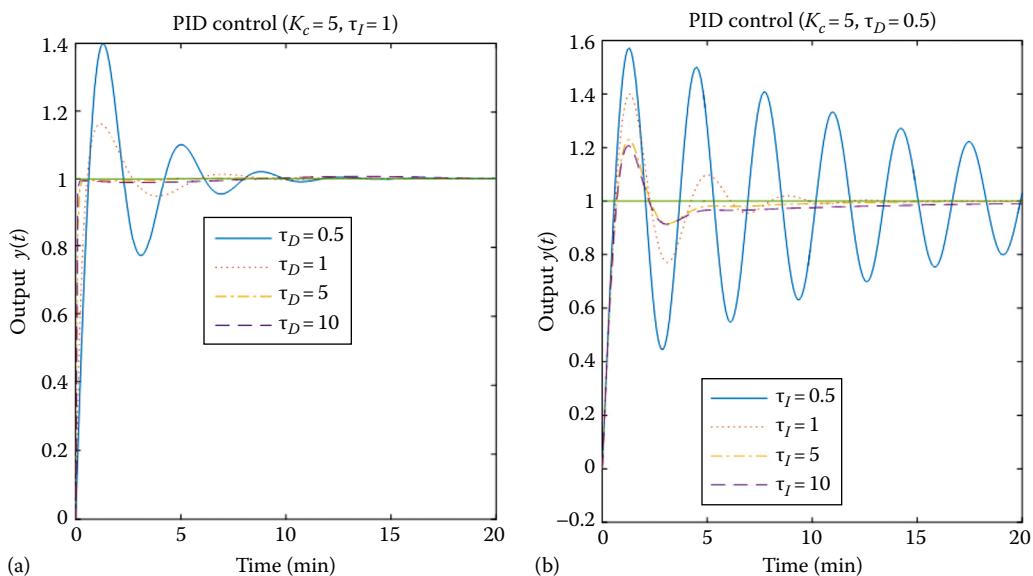


FIGURE 9.18 Effects of derivative time (a) and reset time (b).

```

st = 1+0*t; plot(t,st), hold off
legend('\tau_D=0.5','\tau_D=1','\tau_D=5','\tau_D=10','location','best')
xlabel('Time(min)'), ylabel('Output, y(t)')
title('PID control(Kc=5,\tau_I=1)')

% Constant derivative time (tauD)
tauD = 0.5;
tauI = [0.5 1 5 10]; % try 4 different tauI
subplot(1,2,2)
for i = 1:length(tauI)
    num = Kc*Kp*[tauI(i)*tauD tauI(i) 1]; d1 = tauI(i)*tau^2;
    d2 = 2*tauI(i)*tau*zeta+Kc*Kp*tauI(i)*tauD;
    d3 = tauI(i)*(1+Kc*Kp); d4 = Kc*Kp; den = [d1 d2 d3 d4];
    [y,t] = stepnp(num, den, t0, delt, fint, ms);
    plot(t,y,h(i,:)), hold on
end
st = 1+0*t; plot(t,st), hold off
legend('\tau_I=0.5','\tau_I=1','\tau_I=5','\tau_I=10','location','best')
xlabel('Time(min)'), ylabel('Output, y(t)')
title('PID control(Kc=5,\tau_D=0.5)')

```

The script *tunpid* calls the function *stepnp* to get the step responses. The function *stepnp* calculates step responses of proper single-input, single-output systems.

```

function [y,t] = stepnp(num, den, t0, delt, fint, ms);
% Calculate step response of a proper SISO system
% num : numerator of transfer function
% den : denominator of transfer function
% t0 : time at which unit step input is introduced
% delt : time step
% fint : final response time
% y : step response
% ms: step size

% Partial fraction of (transfer function)*(step input)
% (r: residue vector, p: pole vector, k: constant vector)
[r, p, k] = residue(num, conv(den, [1 0]));
% Set calculation time interval
t = t0 : delt : fint;
% Identify pole multiplicity
for j = 1 : size(p)
    n = 1;
    for i = 1 : size(p)
        if p(j) == p(i)
            if (i ~= j)
                n = n+1;
            end
        end
    end
    mult(:, j) = n;
end
% Step response: use inverse Laplace transform
y = zeros(size(t));
j = 1;
while j <= size(p, 1)
    for i = 1 : mult(:, j)
        y = y + r(j+i-1)*(t-t0).^(i-1).*exp(p(j)*(t-t0))/factorial(i-1);
    end
    j = j + i;
end
y = ms*y;
end

```

The script *tunpid* generates the plots shown in [Figure 9.18](#).

9.5.2 CONTROL OF THE CONTINUOUS STIRRED TANK HEATER

The energy balance for the continuous stirred tank heater with a constant flow rate can be expressed as

$$\rho C_p V \frac{dT}{dt} = w C_p (T_i - T) + Q$$

where

ρ is the fluid density (kg/m^3)

C_p is the fluid heat capacity ($\text{kJ}/(\text{kg} \cdot \text{°C})$)

V is the volume of the heating tank (m^3)

w is the mass flow rate of the fluid (kg/min)

Q is the heat flux from the heating source (kJ/min)

Rearrangement of the energy balance equation yields

$$\frac{dT}{dt} = \frac{1}{\tau} (T_i - T) + \frac{K}{\tau} Q$$

where $\tau = 0(\rho V)/w$ and $K = 1/(wC_p)$. At steady state,

$$Q_s = \frac{1}{K} (T_s - T_{is}) = \frac{1}{K} (T_R - T_{is})$$

where T_R is the set-point. It is assumed that $T_s = T_R$ at steady state. Figure 9.19 shows a continuous stirred-tank heating process.

Since the thermocouple is located at the outflow pipe a short distance downstream, the thermocouple will exhibit a time delay, θ , which is the time required for the output flow to reach the measurement point. Thus, the temperature sensed by the thermocouple can be represented as

$$T_1(t) = T(t - \theta)$$

To handle the time delay, we can use the 1st-order Pade approximation:

$$T_1(s) = e^{-\theta s} T(s) \approx \frac{1 - \theta s/2}{1 + \theta s/2} T(s)$$

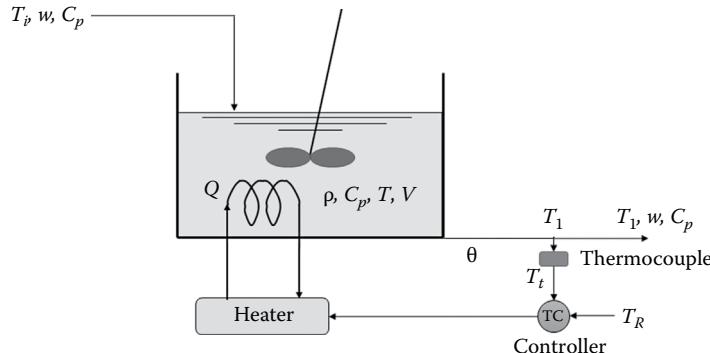


FIGURE 9.19 A continuous stirred-tank heating process.

The inverse Laplace transform on $T_1(s)$ gives

$$\frac{dT_1}{dt} = \frac{2}{\theta} \left(T - T_1 - \frac{\theta}{2} \frac{dT}{dt} \right)$$

If we assume that the thermocouple exhibits a 1st-order dynamics with time constant τ_t and unity gain, the dynamics of the temperature T_t measured by the thermocouple can be represented as

$$\frac{T_t(s)}{T_1(s)} = \frac{1}{\tau_t s + 1} \quad \text{or} \quad \frac{dT_t}{dt} = \frac{1}{\tau_t} (T_1 - T_t)$$

If a PI controller is used, the controller output can be represented in terms of the heat rate Q supplied from the heater:

$$Q(t) = Q_s + K_c (T_R - T_t) + \frac{K_c}{\tau_I} \int_0^t (T_R - T_t) dt$$

where

K_c (kJ/(min · °C)) is the proportional gain of the controller
 τ_I (min) is the reset time

The accumulated control error is given by

$$C_e(t) = \int_0^t (T_R - T_t) dt$$

Differentiation of this equation gives

$$\frac{dC_e(t)}{dt} = T_R - T_t$$

Substitution of these two equations into the controller output equation yields

$$Q(t) = Q_s + \frac{K_c}{\tau_I} C_e(t) + K_c \frac{dC_e(t)}{dt}$$

If a proportional controller is used, the controller output is given by

$$Q(t) = Q_s + K_c (T_R - T_t)$$

Example 9.18: Control of a Stirred-Tank Heating Process

A continuous stirred-tank heating process consists of a stirred tank, heater, and PI controller. The liquid feed with a density of $\rho = 980 \text{ kg/m}^3$ and a heat capacity of $C_p = 1.6 \text{ kJ/(kg} \cdot ^\circ\text{C)}$ flows into the heated tank at a constant mass flow rate of $w = 250 \text{ kg/min}$ and temperature $T_i = 50^\circ\text{C}$. The volume of the tank is $V = 2.5 \text{ m}^3$. It is desired to heat this stream to a higher set-point temperature $T_R = 75^\circ\text{C}$. The outlet temperature is measured by a thermocouple as T_t and the heat flux

Q (kJ/min) supplied by the heater is adjusted by a PI controller with $K_c = 60$ kJ/(min · °C) and $\tau_I = 1.8$ min. The thermocouple exhibits 1st-order dynamics with time constant $\tau_t = 3$ min and unity gain. The time delay between the heating tank and the thermocouple is $\theta = 1.2$ min.

1. The system is initially operating at steady state at a temperature of 75°C (set-point). The inlet temperature T_i is suddenly changed to 30°C at time $t = 10$ min. Assuming no control actions (open-loop, $K_c = 0$), plot the profiles of the fluid temperature in the tank T , the measured temperature T_v and the fluid outlet temperature T_1 .
2. If the controller is engaged (closed loop), plot the profiles of T , T_v and T_1 .
3. If the PI controller is replaced by a proportional (P) controller with $K_c = 60$ kJ/(min · °C), plot the profiles of T , T_v and T_1 .

Solution

The following system of differential equations is to be solved:

$$\frac{dT}{dt} = \frac{1}{\tau} (T_i - T) + \frac{K}{\tau} Q, \quad \frac{dT_1}{dt} = \frac{2}{\theta} \left(T - T_1 - \frac{\theta}{2} \frac{dT}{dt} \right)$$

$$\frac{dT_t}{dt} = \frac{1}{\tau_t} (T_1 - T_t), \quad \frac{dC_e(t)}{dt} = T_R - T_t, \quad Q(t) = Q_s + \frac{K_c}{\tau_I} C_e(t) + K_c \frac{dC_e(t)}{dt}$$

1. The change in the inlet temperature can be represented as

$$T_i(t) = \begin{cases} T_{i1} & t < 10 \\ T_{i2} & t \geq 0 \end{cases}$$

where $T_{i1} = 0$ °C and $T_{i2} = 30$ °C. The function *csthcont* defines the system of differential equations. The script *usecsthcont* employs the built-in function *ode45* to solve the system of differential equations and generates the curves shown in Figure 9.20.

```
csthcont.m
function dz = csthcont(t, z, tau, taut, tauI, K, Kc, theta, Qs, Ti, Tr)
% z(1) = T, z(2) = T1, z(3) = Tt, z(4) = Ce
if t < 10, Ti = 50; else Ti = 30; end
```

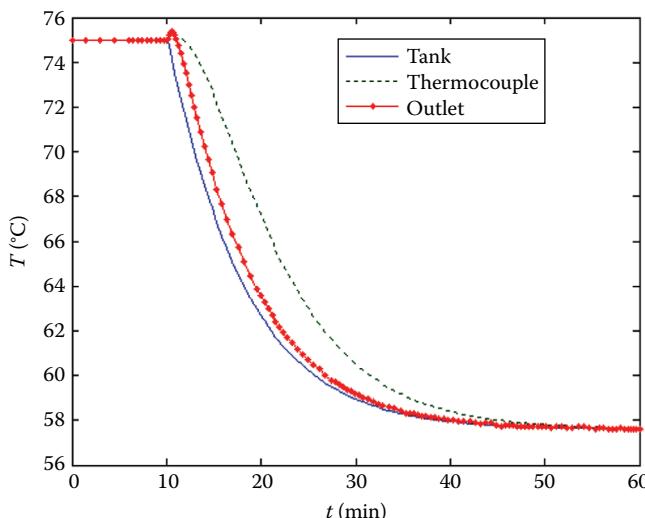


FIGURE 9.20 Closed-loop response of a stirred-tank heating process (P controller).

```

Q = Qs + (Kc/tauI)*z(4) + Kc*(Tr - z(3));
dz(1,1) = (Ti - z(1))/tau + K*Q/tau;
dz(2,1) = 2*(z(1) - z(2) - (theta/2)*dz(1,1))/theta;
dz(3,1) = (z(2) - z(3))/taut;
dz(4,1) = Tr - z(3);
end

usecsthcont.m
% usecsthcont.m
% z(1) = T, z(2) = T1, z(3) = Tt, z(4) = Ce
rho = 980; V = 2.5; Cp = 1.6; w = 250; taut = 3.6; theta = 1.2; Tr = 75;
Ti = 50; tauI = 1.8; K = 1/(Cp*w); tau = rho*V/w; Qs = (Tr-Ti)/K;
z0 = [Tr Tr Tr 0]; % Integral time interval and initial conditions
tv = [0 60]; Kc = 0;
[t z] = ode45(@csthcont, tv, z0, [], tau, taut, tauI, K, Kc, theta, Qs, Ti, Tr);
T = z(:,1); T1 = z(:,2); Tt = z(:,3);
plot(t, T, t, Tt, ':', t, T1, '-.'). xlabel('t(min)'), ylabel('T(C)')
legend('Tank', 'Thermocouple', 'Outlet', 'location', 'best')

```

The script *usecsthcont* produces the outputs shown in Figure 9.21.

```
>> usecsthcont
```

2. In the script *usecsthcont*, set values of Kc and tv as Kc = 60; tv = [0 200]; The script produces the outputs shown in Figure 9.22.

```
>> usecsthcont
```

3. When a proportional controller is used, the heat flux is given by

$$Q(t) = Q_s + K_c (T_R - T_t)$$

The function *csthcontp* defines the system of differential equations, and the script *usecsthcontp* uses the built-in function *ode45* to solve the system of differential equations.

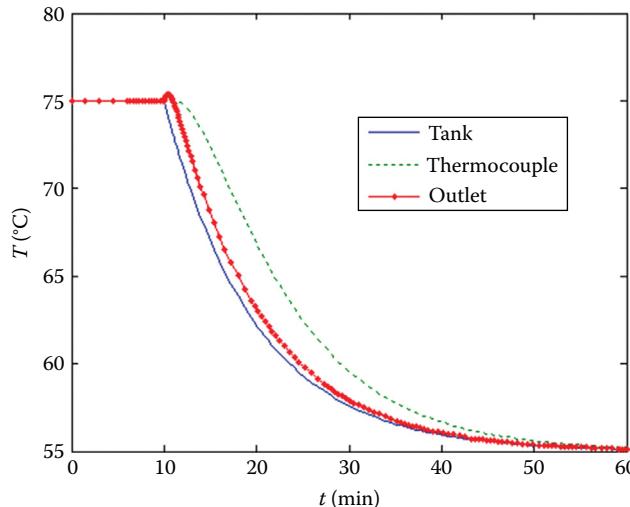


FIGURE 9.21 Open-loop response of a stirred-tank heating process.

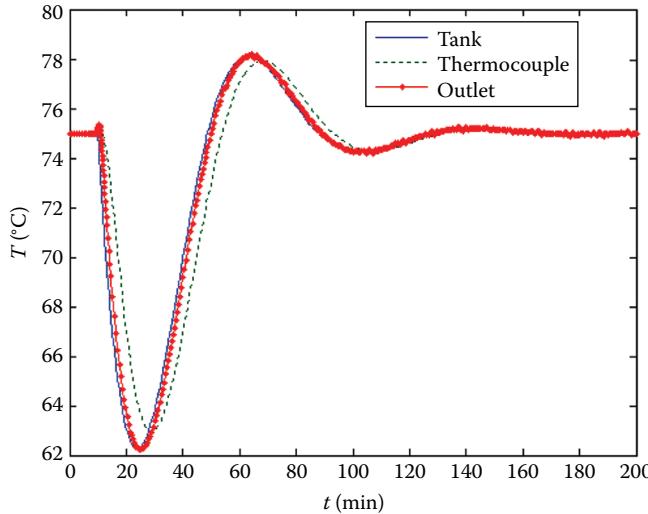


FIGURE 9.22 Closed-loop response of a stirred-tank heating process (PI controller).

```

csthcontp.m
function dz = csthcontp(t,z,tau,taut,K,Kc,theta,Qs,Ti,Tr)
% z(1) = T, z(2) = T1, z(3) = Tt
if t < 10, Ti = 50; else Ti = 30; end
Q = Qs + Kc*(Tr - z(3));
dz(1,1) = (Ti - z(1))/tau + K*Q/tau;
dz(2,1) = 2*(z(1) - z(2) - (theta/2)*dz(1,1))/theta;
dz(3,1) = (z(2) - z(3))/taut;
end

usecsthcontp.m
% usecsthcontp.m
% z(1) = T, z(2) = T1, z(3) = Tt
rho = 980; V = 2.5; Cp = 1.6; w = 250; taut = 3.6; theta = 1.2; Tr = 75;
Ti = 50; Kc = 60;
K = 1/(Cp*w); tau = rho*V/w; Qs = (Tr-Ti)/K;
tv = [0 60]; z0 = [Tr Tr Tr];
[t z] = ode45(@csthcontp, tv, z0, [], tau, taut, K, Kc, theta, Qs, Ti, Tr);
T = z(:,1); T1 = z(:,2); Tt = z(:,3);
plot(t,T,t,Tt,':',t,T1,'.-'), xlabel('t(min)'), ylabel('T(C)')
legend('Tank','Thermocouple','Outlet')

```

9.6 STABILITY OF FEEDBACK CONTROL SYSTEMS

A system is stable if the output response is bounded for any bounded input. The stability of a feedback control system can be analyzed by using the characteristic equation. The overall transfer function of the closed-loop feedback control system is given by

$$C = \frac{G_R G_c G_v G_T G_p}{1 + G_c G_v G_T G_p G_m} R + \frac{G_L}{1 + G_c G_v G_T G_p G_m} L$$

The characteristic equation is

$$1 + G_c G_v G_T G_p G_m = 1 + G_{OL} = 0$$

The roots of the characteristic equation are called the poles of the closed-loop system. The closed-loop control system is stable if all the roots of the characteristic equation have negative real parts. In other words, the closed-loop system is stable if all the poles of the closed-loop transfer function lie in the left half plane. If some roots of the characteristic equation lie on the imaginary axis, the real parts are zero and these roots can be represented as $r_{1,2} = \pm i\omega_u$. The response corresponding to these roots can be expressed as

$$C(s) = \frac{As + B}{s^2 + \omega_u^2} + (\dots) = M \sin(\omega_u t + \phi) + (\dots)$$

The response exhibits sinusoidal oscillations with amplitude M and frequency ω_u . The frequency ω_u is called ultimate frequency and the corresponding period $T_u = (2\pi)/\omega_u$ is called ultimate period.

Since the stability of a closed-loop feedback control system depends on the poles of the system, the stable region of the system can be represented graphically by displaying the poles on the complex plane. All of the poles must lie to the left of the imaginary axis in the complex plane for a system to be stable. A root locus diagram is a trajectory of poles and shows how the poles change when a parameter such as a controller gain changes. By representing the poles for various values of the controller gain K_c on the complex plane, we can see the behavior of the poles as a function of K_c .

A root locus diagram can easily be constructed by using the MATLAB built-in function *rlocus*. The numerator and denominator of the transfer function should be supplied as input arguments to the *rlocus* function.

Example 9.19: Root Locus (1)

Plot the trajectory of poles when the controller gain K_c changes from 1 to 40 for the system with the characteristic equation given by

$$s^3 + 6s^2 + 11s + 6 + 2K_c = 0$$

Solution

The script *rootloc* plots the root locus diagram.

```
% rootloc.m
Kc = [0:1:40];
for i = 1:length(Kc)
    pol = [1 6 11 6+2*Kc(i)]; sol = roots(pol);
    for j = 1:length(sol)
        r(i,j) = sol(j);
    end
end
plot(r,'*'), grid, ylabel('Imaginary part'), xlabel('Real part')
title('Root locus (Kc=1~40)')
```

When the controller gain is greater than 30, some of the poles lie to the right of the imaginary axis in the complex plane and the system becomes unstable. The script *rootloc* generates the locus of poles as shown in [Figure 9.23](#).

```
>> rootloc
```

Example 9.20: Root Locus (2)

Plot the root locus diagram for s system with the characteristic equation given by

$$1 + \frac{K(s+3)}{s^2 + 2s} = 0$$

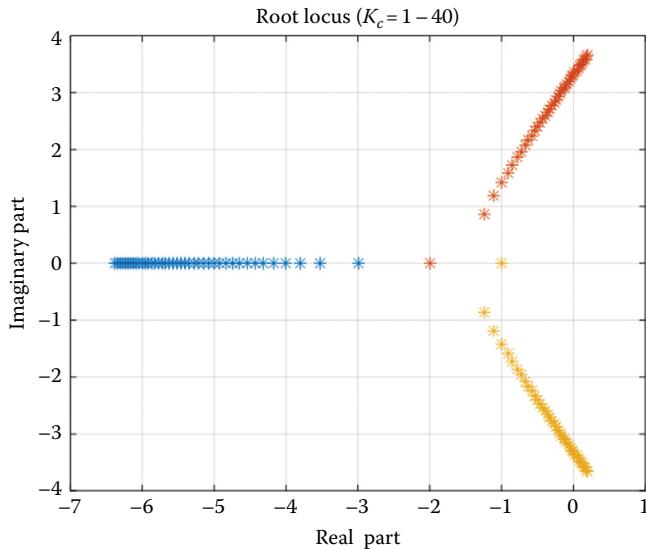


FIGURE 9.23 Locus of poles when the controller gain changes.

Solution

The numerator and denominator of the open-loop transfer function are $s + 3$ and $s^2 + 2s$, respectively. These are supplied to the built-in function *rlocus* as input arguments. The function sets the range of K automatically to construct the root locus diagram shown in Figure 9.24.

```
>> num = [1 3]; den = [1 2 0]; rlocus(num, den);
>> grid, title('Root locus'), xlabel('Real axis'), ylabel('Imaginary axis')
```

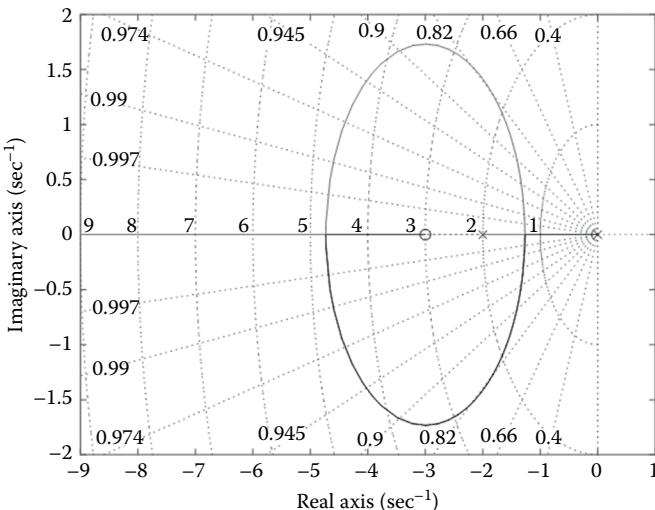


FIGURE 9.24 Root locus.

9.7 FREQUENCY RESPONSE ANALYSIS

The response of a stable system at large times is characterized by its amplitude and phase shift when the input is a sinusoidal wave. Consider a stable process whose transfer function is given by

$$G(s) = \frac{Y(s)}{X(s)} = \frac{N(s)}{(s - r_1)(s - r_2) \cdots (s - r_n)}$$

The response to a general sinusoidal input, $x(t) = A \sin(\omega t)$, or $X(s) = (A\omega)/(s^2 + \omega^2)$, can be expressed as

$$Y(t) = a_1 e^{r_1 t} + a_2 e^{r_2 t} + \cdots + a_n e^{r_n t} + C \cos \omega t + \frac{D}{\omega} \sin \omega t$$

where C and D are constants. Since the system is stable, the real parts of all the roots r_1, r_2, \dots, r_n are negative. Thus if the sinusoidal input is continued for a long time, the exponential terms will decay away and the time response becomes a purely sinusoidal function represented by cosine and sine terms. The remaining cosine and sine terms can be combined to yield

$$Y(t) = AI \cos \omega t + \frac{AR\omega}{\omega} \sin \omega t = A\sqrt{R^2 + I^2} \sin(\omega t + \phi) = \hat{A} \sin(\omega t + \phi)$$

where $\hat{A} = A\sqrt{R^2 + I^2}$ and $\phi = \tan^{-1}(I/R)$. The amplitude ratio (AR) is given by

$$\text{AR} = \frac{\hat{A}}{A} = \frac{A\sqrt{R^2 + I^2}}{A} = \sqrt{R^2 + I^2} = |G(i\omega)|$$

We can see that the amplitude ratio is the magnitude of the complex number obtained by setting $s = i\omega$ in $G(s)$.

9.7.1 BODE DIAGRAM

The Bode diagram is a graphical representation of the frequency response characteristics of a transfer function model. The Bode diagram consists of a log-log plot of the amplitude AR versus ω (radians/time) and a semilog plot of the phase angle ϕ versus ω . The built-in function *bode* constructs the Bode plot and calculates the amplitude ratio and the phase angle.

9.7.2 NYQUIST DIAGRAM

The Nyquist diagram, an alternative representation of frequency response information, is a graphical representation of the real and imaginary parts of $G(i\omega)$ on the s plane with ω as the parameter. Because a complex number can be put in polar coordinates, the Nyquist diagram is also referred to as the polar plot of $G(i\omega)$. The Bode diagram is easier to interpret, while the Nyquist diagram is more widely used in multiloop or multivariable analysis. The MATLAB built-in functions *nyquist* and *polar* can both be used to construct the Nyquist diagram.

9.7.3 NICHOLS CHART

The Nichols chart is a plot of the logarithmic magnitude against the phase lag. In MATLAB, the Nichols chart can be constructed by the built-in function *nichols*.

Example 9.21: Bode Diagram of a 2nd-Order Process

Plot the Bode diagram of a 2nd order process $G(s) = 1/(2.25s^2 + 3\zeta s + 1)$ when $\zeta = 0, 0.25, 0.5, 0.75$, and 1 .

Solution

The script *bodezeta* generates the Bode plot for each value of ζ .

```
% bodezeta.m
w = logspace(-2, 1, 300); zeta = [0:0.25:1]; num = 1;
for i = 1:length(zeta),
    den = [2.25 3*zeta(i) 1]; [ar, phase] = bode(num, den, w);
    lar = 20*log10(ar); semilogx(w, lar); hold on
end
xlabel('w(rad/min)'), ylabel('AR'), title('Response of 2nd-order process')
text(0.8, 20, 'zeta=0'), text(0.4, -10, 'zeta=1'), grid
```

The script *bodezeta* produces the curves shown in [Figure 9.25](#).

Example 9.22: Bode Diagram of a 3rd-Order Process

Plot the Bode diagram of a 3rd-order process with time delay:

$$G(s) = \frac{(0.4s+1)e^{-0.2s}}{(0.3s+1)(s+1)^2}$$

Solution

The script *delaybode* generates the Bode plot for the 3rd-order process with time delay. In the script, the option 'iodelay' is used to handle the time delay.

```
% delaybode.m
num = [0.4 1]; den = conv([0.3 1], conv([1 1], [1 1]));
G = tf(num, den, 'iodelay', 0.2); [mag, phase, w] = bode(G);
```

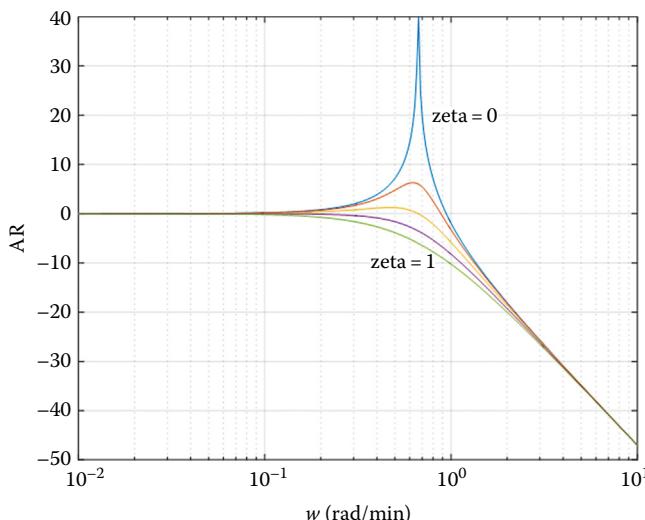


FIGURE 9.25 Bode diagram for a 2nd-order process.

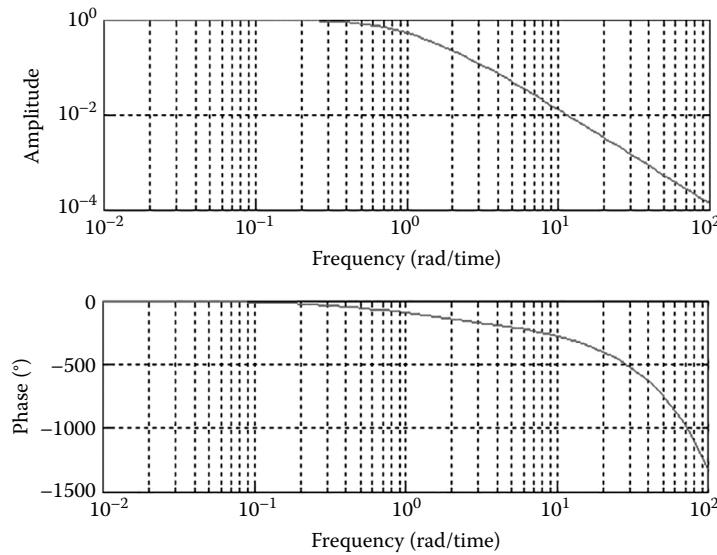


FIGURE 9.26 Bode diagram of a process with time delay.

```
subplot(2,1,1), loglog(w, squeeze(mag)), grid
ylabel('Amplitude'), xlabel('Frequency (rad/time)')
subplot(2,1,2), semilogx(w, squeeze(phase)), grid
ylabel('Phase (deg)'), xlabel('Frequency (rad/time)')
```

The script *delaybode* produces the Bode plot shown in Figure 9.26:

```
>> delaybode
```

Example 9.23: Nyquist Diagram of a 2nd-Order Process

Plot the Nyquist diagram for a 2nd-order process whose transfer function is

$$G(s) = \frac{2}{(10s+1)(2.5s+1)}$$

Solution

Both the built-in functions *polar* and *nyquist* can be used. The following commands use the function *polar* to generate the polar plot shown in Figure 9.27.

```
>> w = logspace(-2, 1, 300); num = 2; den = conv([10 1], [2.5 1]);
>> [x, p] = bode(num, den, w); polar((pi/180)*p, x)
>> title('Polar plot of a 2^{nd}-order plot')
```

The following commands use the function *nyquist* to plot the Nyquist diagram shown in Figure 9.28.

```
>> w = logspace(-2, 1, 300); num = 2; den = conv([10 1], [2.5 1]);
>> nyquist(num, den)
```

The Nyquist diagram can also be constructed by plotting the outputs obtained from the function *nyquist*.

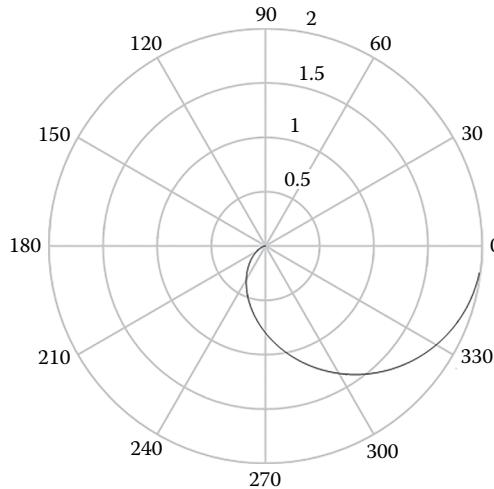


FIGURE 9.27 Nyquist plot of a 2nd-order process by polar function.

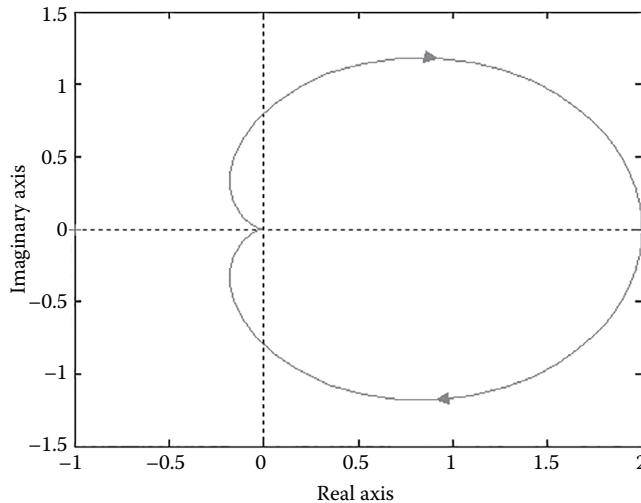


FIGURE 9.28 Nyquist plot of a 2nd-order process by nyquist function.

Example 9.24: Nyquist Diagram of a Time Delay

Plot the Nyquist diagram for a 1st-order process with a time delay whose transfer function is

$$G(s) = \frac{12.76e^{-s}}{5s+1}$$

Solution

The following commands generate the desired Nyquist diagram shown in [Figure 9.29](#).

```
>> G = tf(12.76, [5 1]); set(G, 'iodelay', 1); nyquist(G)
>> xlabel('Real axis'), ylabel('Imaginary axis')
>> title('Nyquist diagram of a time delay')
```

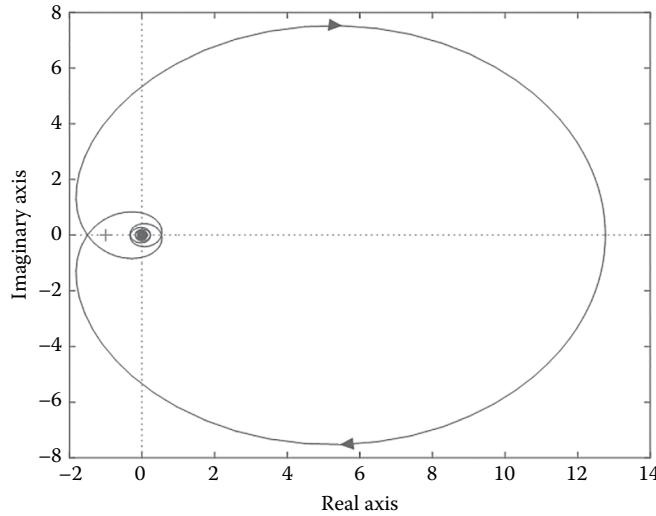


FIGURE 9.29 Nyquist diagram of a process with time delay.

Example 9.25: Nichols Chart

Plot the Nichols chart for a 2nd-order process whose transfer function is

$$G(s) = \frac{2}{(10s+1)(2.5s+1)}.$$

Solution

The following commands generate the desired Nichols chart shown in Figure 9.30.

```
>> num = 2; den = conv([10 1], [2.5 1]); w = logspace(-2, 1, 300);
>> nichols(num, den, w); ngrid
```

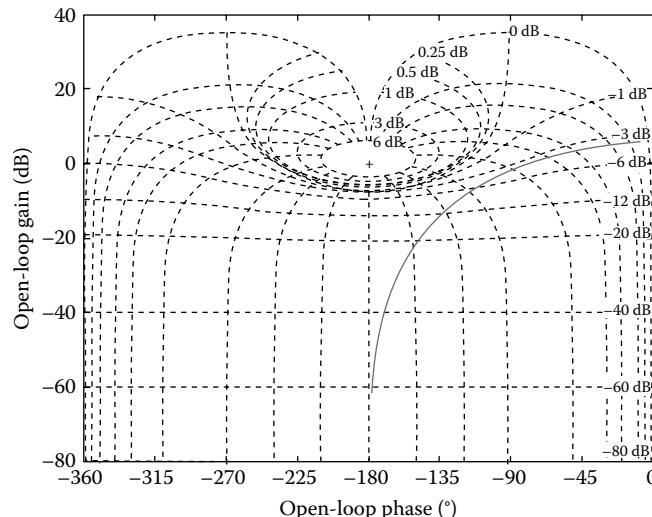


FIGURE 9.30 Nichols chart of a 2nd-order process.

9.7.4 GAIN AND PHASE MARGINS

Gain margin (GM) and phase margin (PM) provide quantitative measures of relative stability that indicate how close the system is to becoming unstable. The ultimate controller gain K_{cu} is the gain at which the system is at marginal stability. Thus, when the controller gain K_c is adjusted close to K_{cu} , the control system is close to an unstable state. Let AR_c be the value of the open-loop amplitude ratio at the ultimate frequency ω_c when $\phi = -180^\circ$, and let ϕ_g be the phase angle at the frequency ω_g when $AR = 1$. The GM and PM are defined as, respectively,

$$GM = \frac{1}{AR_c}, \quad PM = 180 + \phi_g$$

In practice, a controller is tuned so that a GM lies between 1.7 and 4.0 and a PM lies between 30° and 45° . The smaller the values of GM and PM, the closer the system to the point of marginal stability, and the closed-loop system exhibits very oscillatory dynamics. The controller gain can be set by the ultimate gain K_{cu} and GM as

$$K_c = \frac{K_{cu}}{GM}$$

In MATLAB, GM and PM are obtained from the built-in function *margin*.

Example 9.26: Ultimate Gain

The characteristic equation of a feedback control system using a proportional controller is given by

$$1 + K_c \frac{0.8e^{-2s}}{5s+1} = 0$$

where K_c is the controller gain. Find the ultimate gain.

Solution

The following commands produce the desired outputs.

```
>> G = tf(0.8, [5 1]); delay = 2; [m, p, w] = bode(G); Mag = m(1, :);
>> Phase = p(1, :) - ((180/pi)*delay*w');
>> [Gm, Pm, Wcg, Wcp] = margin(Mag, Phase, w)
Gm =
    5.7173
Pm =
    Inf
Wcg =
    0.8922
Wcp =
    NaN
```

We can see that $K_{cu} = 5.7173$ and $\omega_u = 0.8922$. The following commands generate a plot showing GM and PM (see Figure 9.31).

```
>> G = tf(0.8, [5 1]); theta = 2; set(G, 'iodelay', theta); margin(G)
```

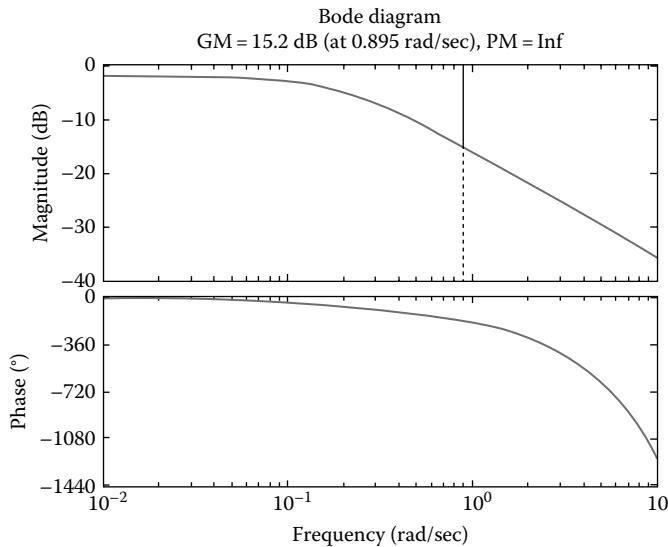


FIGURE 9.31 GM and PM of a process with time delay.

PROBLEMS

- 9.1** Expand $F(s) = (s^2 + 2s + 3)/(s^3 + 3s^2 + 3s + 1)$ into partial fractions.
- 9.2** Two blocks connected in series can be represented as $C = G_1 G_2 A$. Synthesize these two blocks. The transfer function of each block is given by $G_1 = 2/(s + 1)$ and $G_2 = (2s + 1)/(s^2 + 3s + 2)$.
- 9.3** Two blocks connected in parallel can be represented as $C = (G_1 + G_2)A$. Synthesize these two blocks. The transfer function of each block is given by $G_1 = 2/(s + 1)$ and $G_2 = (2s + 1)/(s^2 + 3s + 2)$.
- 9.4** The state-space representation of a process is given by

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & -3 \\ 1 & 0 & -7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} u, \quad y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Find the transfer function for the process.

- 9.5** Find the impulse response of a 1st-order process with the transfer function given by $3/(2s + 1)$ during the time interval of $[0, 10]$.
- 9.6** The transfer function of a 1st-order process is given by $3/(2s + 1)$. Find the response curve of the process for the linear input $x = 1.2t$ during the time interval of $[0, 10]$.
- 9.7** In a 2nd-order process, the time constant and the process gain are given by $\tau = 2$ and $k = 3$, respectively. A step change of the magnitude $\Delta m = 4$ is introduced to the process input. Plot the dimensionless response $y(t)/(k\Delta m)$ as a function of time t ($0 \leq t \leq 40$) for each value of the decay ratio $\zeta = 0.1, 0.4, 0.7$, and 1 .
- 9.8** In a 2nd-order process, the time constant and the process gain are given by $\tau = 2$ and $k = 3$, respectively. A step change of the magnitude $\Delta m = 4$ is introduced to the process input. Plot the dimensionless response $y(t)/(k\Delta m)$ as a function of time t ($0 \leq t \leq 40$) for each value of the decay ratio $\zeta = 1, 1.5, 2.0$, and 2.5 .
- 9.9** The transfer function of a 2nd-order process is given by $1/(\tau^2 s^2 + 2\tau\zeta s + 1)$ where $\tau = 0.5$. Plot the impulse response curve for each value of the decay ratio $\zeta = 0.5, 1.0$, and 1.5 during the time interval of $[0, 10]$.

- 9.10** The transfer function of a lead/lag element is given by $(\tau s + 1)/(s + 1)$. Plot the unit step response curve for each value of $\tau = 0, 0.5, 1.0, 1.5$, and 2 during the time interval of $[0, 10]$.
- 9.11** A proportional (P) controller is used in a feedback control system. The process transfer function is given by $G_p(s) = 5/(5s + 1)$, the gain of the control valve is $K_v = 0.01$, and the gain of the sensor/transducer is $K_m = 20$. Use the Simulink to find the step response curve to a unit step change in the set-point. The controller gain K_c is 10 .
- 9.12** A PI controller is used to control a 1st-order process $G = 0.8/(5s + 1)$. The transfer function of the controller is $G_c(s) = K_c(1 + (1/\tau_I s))$ where $K_c = 1$ and $\tau_I = 10$. Find the step response curve to a unit step change in the set-point.
- 9.13** A proportional–integral (PI) controller is used in a liquid-level control system. The process transfer function is $G(s) = 2/(4.8s + 1)$, the gain of the control valve is 0.08 and the reset time of the PI controller is $\tau_I = 2$. Use the Simulink to find the step response curve to a unit step change in the inlet flow rate for each value of the controller gain $K_c = 3, 10, 20, 60$. To construct the PI controller block in the Simulink, select the PID block from the Simulink → Continuous submenu in the Simulink Library Browser and drag the block to the block diagram in the editor window. Double-click the PID block to set the controller parameters. Set both the filter coefficient N and the derivative time to zeros.
- 9.14** Figure P9.14 shows two liquid-level tanks connected in an interacting fashion. Assuming constant fluid density, the material balance for each tank gives

$$A_1 \frac{dh_1}{dt} = q_i - C_1 \sqrt{h_1 - h_2}, \quad A_2 \frac{dh_2}{dt} = C_1 \sqrt{h_1 - h_2} - C_2 \sqrt{h_2}$$

where

A_1 and A_2 are cross sectional areas of tank 1 and tank 2, respectively
 C_1 and C_2 are parameters

The manipulated variable is the flow rate q_i (ft^3/min) to the first tank. When a PI controller is used, the inlet flow rate to the first tank is given by

$$\frac{q_i}{q_m} = r_m + K_c e(t) + \frac{K_c}{\tau_I} \int_0^t e(t) dt, \quad e(t) = h_R - h_2$$

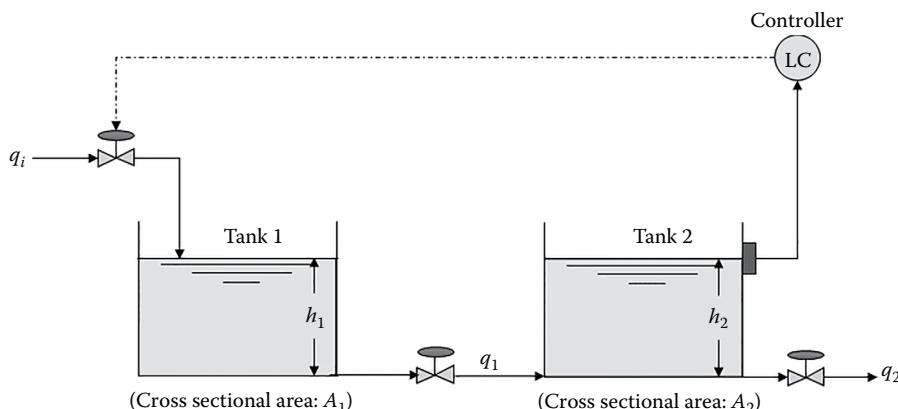


FIGURE P9.14 Two tanks connected in an interacting fashion.

where

K_c (ft⁻¹) is the controller gain

τ_I (min) is the reset time

q_m is the maximum possible flow rate (i.e., $0 \leq q_i \leq q_m$)

r_m is the fractional flow rate when there is no control action (i.e., $q_i = r_m q_m$)

h_R is the set-point for the second tank level

The accumulated control error can be represented as

$$C_e(t) = \int_0^t e(t) dt = \int_0^t (h_R - h_2) dt$$

Differentiation of this equation yields

$$\frac{dC_e(t)}{dt} = e(t) = h_R - h_2$$

Thus, the controller equation can be expressed as

$$\frac{q_i}{q_m} = r_m + \frac{K_c}{\tau_I} C_e(t) + K_c \frac{dC_e(t)}{dt}$$

If a P controller is used, the controller equation will be

$$\frac{q_i}{q_m} = r_m + K_c (h_R + h_2)$$

The following information and data apply: $A_1 = A_2$, $C_1 = 61.8$ ft^{2.5}/min, $C_2 = 30.9$ ft^{2.5}/min, $h_R = 6$ ft, $r_m = 0.4$, $q_m = 480$ ft³/min, $d = 12$ ft (d : tank diameter).

1. Plot the liquid level of each tank versus time when there is no control action.
2. A P controller is used to control the liquid level. Plot the liquid level of the second tank versus time when the controller gain is $K_c = 2$ and $K_c = 40$.
3. A PI controller is used to control the liquid level. Plot the liquid level of each tank versus time when the controller gain is $K_c = 0.2$ and the reset time is $\tau_I = 20$ min.

9.15 Determine the stability of a system whose characteristic equation is given by

$$s^4 + 5s^3 + 3s^2 + 1 = 0.$$

9.16 The transfer function of a PID controller can be expressed as

$$G_c(s) = \frac{M(s)}{E(s)} = K_p + \frac{K_I}{s} + K_D s = \frac{K_D s^2 + K_p s + K_I}{s}$$

where $K_c = K_p$, $K_c \tau_D = K_D$ and $(K_c / \tau_I) = K_I$. Plot the root locus diagram for arbitrary tuning parameters when the process transfer function is given by $G_p(s) = 1/s^2$.

9.17 Plot the Bode diagram for a 2nd-order process whose transfer function is given by

$$G(s) = \frac{2}{(10s+1)(2.5s+1)}.$$

- 9.18** Plot the Nyquist diagram for a 3rd-order system whose transfer function is given by $G(s) = 1/(s^3 + 2s^2 + 2s + 1)$.
- 9.19** A P controller is used in a feedback control system whose characteristic equation is given by

$$1 + \frac{K_c}{(s+1)(s+2)(s+3)} = 0$$

Find the ultimate gain.

10 Optimization

Optimization is the computational process of minimizing or maximizing an objective function while satisfying the prevailing constraints. Optimization has been one of the most important engineering tools to address high energy cost, global competition in product price and quality, and stringent environmental regulations. In the chemical engineering industry, application of optimization techniques in production, process control, scheduling, inventory control, and transportation has resulted in significant cost savings and improved operational environments. In order for chemical engineers to apply optimization at their working place, they must understand both the basic algorithms and computational procedures.

In this chapter, we focus on optimization algorithms and MATLAB® programs that implement optimization techniques on practical problems. Presentation of theoretical concepts and derivations of optimization formula as well as rigorous optimization proofs are omitted. Introduction of each optimization algorithm is followed by MATLAB examples to illustrate applications and to provide the reader with an understanding of how the optimization technique works.

The main objective of this chapter is to provide readers with various optimization techniques and corresponding MATLAB programs that have already been found, or can potentially be used, for academic or industrial applications. The MATLAB programs can be used in undergraduate or graduate courses on optimization. Researchers and practicing engineers in the field of process engineering can customize and apply these MATLAB programs in the design, simulation, and operation of chemical processes.

10.1 UNCONSTRAINED OPTIMIZATION

10.1.1 FIBONACCI METHOD

The Fibonacci method is most convenient when the interval containing the extremum (minimum or maximum) point or the interval of uncertainty is to be reduced to a given value in the least number of trials or function evaluations. If the Fibonacci numbers are represented as $F_0, F_1, F_2, \dots, F_n, \dots$, the sequence can be generated using

$$F_0 = 1, \quad F_1 = 1, \quad F_i = F_{i-1} + F_{i-2} \quad (i = 2, \dots, n)$$

Figure 10.1 shows the interval reduction procedure, which can be described as

$$I_1 = I_2 + I_3, \quad I_2 = I_3 + I_4, \dots, \quad I_k = I_{k+1} + I_{k+2}, \dots, \quad I_{n-2} = I_{n-1} + I_n, \quad I_{n-1} = 2I_n$$

Using the definition of the Fibonacci number, we have

$$I_1 = F_n I_n, \quad I_2 = F_{n-1} I_n, \quad I_{n-k} = F_{k+1} I_n$$

The minimization algorithm based on the Fibonacci method is given below:

1. Specify two points x_1 and x_4 representing the initial interval $I_1 = |x_4 - x_1|$.
2. Specify the number of interval reductions n . If a desired accuracy ϵ is given, n can be defined as the smallest value such that $\frac{1}{F_n} < \epsilon$.

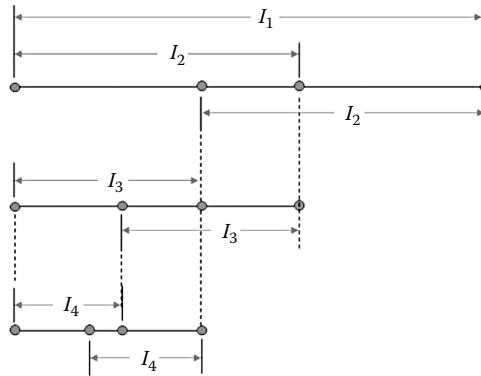


FIGURE 10.1 Interval reduction.

3. Calculate $v = \frac{\sqrt{5}-1}{2}$, $w = \frac{1-\sqrt{5}}{1+\sqrt{5}}$, $\alpha = \frac{v(1-w^n)}{1-w^{n+1}}$.

4. Determine intermediate points $x_3 = \alpha x_4 + (1-\alpha)x_1$ and calculate $f_3 = f(x_3)$.

5. for $i = 1: n - 1$

 if $i = n - 1$, $x_2 = 0.01x_1 + 0.99x_3$

 else $x_2 = \alpha x_1 + (1-\alpha)x_4$

 endif

 calculate $f_2 = f(x_2)$.

 if $f_2 < f_3$, $x_4 = x_3$, $x_3 = x_2$, $f_3 = f_2$

 else $x_1 = x_4$, $x_4 = x_2$

 endif

$$\alpha = \frac{v(1-w^{n-i})}{1-w^{n-i+1}}$$

end for

The MATLAB function *fbnopt* implements the Fibonacci search algorithm. The basic syntax is

```
[x, f, fint] = fbnopt(objfun, a, b, n)
```

where *objfun* is the objective function, *a* and *b* are the initial interval points, *n* is the number of reductions, *x* is the resultant optimum point, *f* is the function value at the optimum point, and *fint* is the length of the final interval.

```
function [x, f, fint] = fbnopt(objfun, a, b, n)
% fbnopt.m: 1-dimensional Fibonacci search method
% inputs:
%     objfun: objective function
%     a, b: initial interval points
%     n: number of reduction
% output:
%     fint: size of final interval
%     x: optimum point
%     f: function value at the optimum point
```

```
% Sample run:
%     a = 0; b = 50; n = 20; fobj= @(x) -870*x + 102*x^2 - 5*x^3;
%     [x, f, fint] = fbnopt(fobj, a, b, n);

v = (sqrt(5)-1)/2; w = (1-sqrt(5))/(1+sqrt(5)); x1 = a; x4 = b; alpha =
v*(1-w^n)/(1-w^(n+1));
x3 = alpha*x4 + (1-alpha)*x1; f3 = objfun(x3);
for k = 1: n - 1
    if k == n - 1
        x2 = 0.01*x1 + 0.99*x3;
    else
        x2 = alpha*x1 + (1-alpha)*x4;
    end
    f2 = objfun(x2);
    if (f2 < f3)
        x4 = x3; x3 = x2; f3 = f2;
    else
        x1 = x4; x4 = x2; f4 = f2;
    end
    alpha = v*(1- w^(n - k)) / (1-w^(n - k + 1));
end
x = x3; f = f3; fint = abs(x1 - x4);
```

Example 10.1: Fibonacci Search

Use the Fibonacci search method to find the minimum of $f(x) = 2x^2 \sin(1.5x) - 4x^2 + 3x - 1$. The initial interval can be specified as $[-8, 8]$. Let the number of interval reductions be $n = 20$.

Solution

```
>> a = -8; b = 8; n = 20; fobj = @(x) 2*x^2*sin(1.5*x) - 4*x^2 + 3*x-1;
>> [x, f, fint] = fbnopt(fobj, a, b, n)
x =
-5.7256
f =
-197.9705
fint =
0.0015
```

10.1.2 GOLDEN SECTION METHOD

From the interval reduction procedure shown in [Figure 10.1](#),

$$I_1 = I_2 + I_3, \quad I_2 = I_3 + I_4, \dots$$

As the number of iterations is increased, the ratio of the Fibonacci numbers F_{n-1}/F_n reaches the limit $\tau = (\sqrt{5} - 1)/2 = 0.61803$. The limit τ is called the golden ratio. Now we wish to maintain

$$\frac{I_2}{I_1} = \frac{I_3}{I_2} = \frac{I_4}{I_3} = \dots = \tau$$

From these relations, we have

$$I_2 = \tau I_1, \quad I_3 = \tau I_2 = \tau^2 I_1$$

Substituting these equations into $I_1 = I_2 + I_3$, we have

$$1 = \tau + \tau^2$$

The positive root of this equation is given by

$$\tau = \frac{\sqrt{5} - 1}{2} = 0.61803$$

which is the golden ratio.

The interval reduction scheme by the golden section method follows the procedure used in the Fibonacci algorithm. The golden section algorithm is given below:

1. Specify the initial point x_1 and the step size h . Evaluate $f_1 = f(x_1)$.
2. Let $x_2 = x_1 + h$ and evaluate $f_2 = f(x_2)$.
3. If $f_2 > f_1$, interchange points 1 and 2. Set $h = -h$.
4. Set $h = \frac{h}{\tau}$, $x_4 = x_2 + h$ and evaluate $f_4 = f(x_4)$.
5. If $f_4 > f_2$, go to step 7.
6. Let $x_1 = x_2$, $x_2 = x_4$ and go to step 4.
7. Set $x_3 = \tau x_4 + (1 - \tau)x_1$ and evaluate $f_3 = f(x_3)$.
8. If $f_2 < f_3$, set $x_4 = x_1$ and $x_1 = x_3$. Else set $x_1 = x_2$, $x_2 = x_3$, and $f_2 = f_3$.
9. If the stopping criterion is not satisfied, go to step 7.

The MATLAB function *gsopt* implements the golden section search algorithm. The basic syntax is

```
[x, f, n] = gsopt(objfun, x1, h, crit)
```

where *objfun* is the objective function, *x1* is the initial point, *h* is the step size, *crit* is the stopping criterion, *x* is the resultant optimum point, *f* is the function value at the optimum point, and *n* is the length of the final interval.

```
function [x, f, n] = gsopt(objfun, x1, h, crit)
% gsopt.m: golden section search method
% input:
%   objfun: objective function
%   x1: initial point
%   h: step size
%   crit: stopping criterion
% output:
%   x: optimal point
%   f: function value at the optimal point
%   n: number of function evaluations
% sample run:
%   crit=1e-6;, x1=1; h=0.1;
%   fun = @(x) 85*(1-x^3)^2 + (1-x^2) + 3*(1-x)^2;
%   [x, f, n] = gsopt(fun, x1, h, crit)

% Initialization
tau = (sqrt(5) - 1)/2; n = 0; f1 = objfun(x1); n = n+1;
x2 = x1 + h; f2 = objfun(x2); n = n+1;
```

```
% Golden section search
if f2 > f1
    temp = x1; x1 = x2; x2 = temp; temp = f1; f1 = f2; f2 = temp; h = -h;
end
while x1 < 1e50
    h = h/tau; x4 = x2 + h; f4 = objfun(x4); n = n+1;
    if f4 > f2, break; end
    f1 = f2; x1 = x2; f2 = f4; x2 = x4;
end
fold = (f1 + f2 + f4) / 3; ind = 0;
while x1 < 1e50
    if abs(x4-x1) < crit, break; end
    x3 = tau*x4 + (1-tau)*x1; f3 = objfun(x3); n = n+1;
    if f2 < f3
        x4 = x1; x1 = x3; f4 = f1; f1 = f3;
    else
        x1 = x2; x2 = x3; f1 = f2; f2 = f3;
    end
    fpr = (f1 + f2 + f4) / 3;
    if abs(fpr - fold) < crit
        ind = ind + 1;
        if ind == 2, break; end
    else
        ind = 0;
    end
    fold = fpr;
end
x = x2; f = f2;
end
```

Example 10.2: Golden Section Method

Use the golden section search to find the minimum of $f(x) = 2x^2\sin(1.5x) - 4x^2 + 3x - 1$. The initial point can be specified as $x_1 = -5$. Let the step size be $h = 0.1$ and the stopping criterion be 1×10^{-6} .

Solution

```
>> crit = 1e-6; x1 = -5; h = 0.1;
>> fobj = @(x) 2*x^2*sin(1.5*x) - 4*x^2 + 3*x-1;
>> [x, f, n] = gsopt(fobj, x1, h, crit)
x =
    -5.7248
f =
    -197.9705
n =
    25
```

Note that the given objective function has many local minimum values as shown in [Figure 10.2](#). Thus, different initial point gives different results. For example, $x_1 = 1$ gives $x_{\min} = 3.7939$ and $f_{\min} = -63.265$, while $x_1 = 6$ gives $x_{\min} = 7.6665$ and $f_{\min} = -316.0254$ as shown below.

```
>> x1 = 1; [x, f, n] = gsopt(fobj, x1, h, crit)
x =
    3.7939
f =
    -63.2650
n =
    28
```

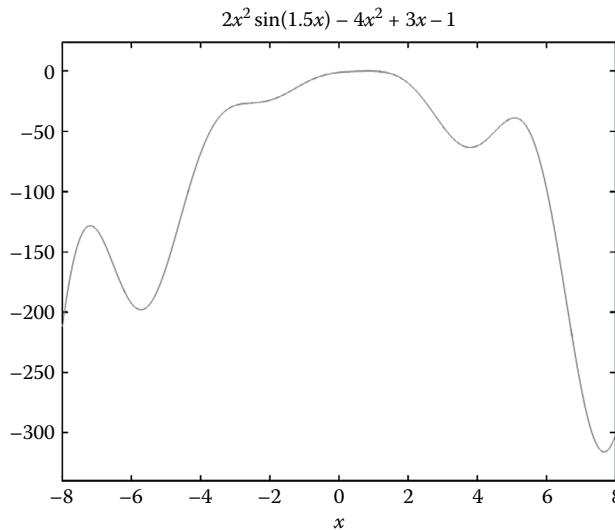


FIGURE 10.2 Plot of $f(x) = 2x^2 \sin(1.5x) - 4x^2 + 3x - 1$.

```

>> x1 = 6; [x, f, n] = gsopt(fobj, x1, h, crit)
x =
    7.6665
f =
   -316.0254
n =
    28

```

10.1.3 BRENT'S QUADRATIC FIT METHOD

Suppose that initial three data points are given as (x_1, f_1) , (x_2, f_2) , (x_3, f_3) where $x_1 < x_2 < x_3$ and $f_2 \leq \min(f_1, f_3)$. A quadratic function of a form $p(x) = a + bx + cx^2$ can be fitted through these points. Using the Lagrange polynomial, the function $p(x)$ can be expressed as

$$p(x) = f_1 \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + f_2 \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + f_3 \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

The minimum point $x^* = x_4$ can be found by setting $dp(x)/dx = 0$, which gives

$$x^* = x_4 = \frac{BCf_1(x_2 + x_3) + ACf_2(x_1 + x_3) + ABf_3(x_1 + x_2)}{2(BCf_1 + ACf_2 + ABf_3)}$$

$$A = (x_1 - x_2)(x_1 - x_3), \quad B = (x_2 - x_1)(x_2 - x_3), \quad C = (x_3 - x_1)(x_3 - x_2)$$

A new interval of uncertainty $(x_{1new}, x_{2new}, x_{3new})$ is determined by comparing f_2 with $f_4 = f(x_4)$:

$$(x_{1new}, x_{2new}, x_{3new}) = \begin{cases} (x_1, x_2, x_4) : x_4 > x_2, f_4 \geq f_2 \\ (x_2, x_4, x_3) : x_4 > x_2, f_4 < f_2 \\ (x_4, x_2, x_3) : x_4 < x_2, f_4 \geq f_2 \\ (x_1, x_4, x_2) : x_4 < x_2, f_4 < f_2 \end{cases}$$

Brent's quadratic fit method starts with bracketing of an interval that has the minimum. At any step, five points (a, b, x, v, w) are considered. Points a and b bracket the minimum, x is the point that gives the least function value, w is the point that gives the second least function value, and v is the previous value of w . Quadratic fitting is tried for x, v , and w , and the quadratic minimum point s is given as

$$s = x - \frac{1}{2} \frac{(x-w)^2 \{f(x) - f(v)\} - (x-v)^2 \{f(x) - f(w)\}}{(x-w)\{f(x) - f(v)\} - (x-v)\{f(x) - f(w)\}}$$

Suppose that the estimated points are equidistant with an equal distance h . Then the estimated points w, x, v and corresponding function values can be expressed as

$$w = x_0, \quad x = x_1 = w + h = x_0 + h, \quad v = x_2 = x + h = x_0 + 2h$$

$$f_0 = f(w) = f(x_0), \quad f_1 = f(x) = f(x_1), \quad f_2 = f(v) = f(x_2)$$

Substitution of these relations to the above equation yields

$$s = x_0 + \frac{h}{2} \left(\frac{4f_1 - 3f_0 - f_2}{2f_1 - f_0 - f_2} \right)$$

Brent's quadratic fit algorithm for minimum is given below¹:

1. Specify three points a, b (form the interval), and x (gives the least function value).
2. Initialize w and v at x .
3. If x, w , and v are all distinct, go to step 5.
4. Find the point u using the golden section search method for the larger of the two intervals $x - a$ or $x - b$. Go to step 7.
5. Perform quadratic fitting for x, w , and v , and find the minimum point u .
6. Adjust u into the larger interval of $x - a$ or $x - b$.
7. Calculate $f_u = f(u)$.
8. Find new values for a, b, x, w , and v .
9. Check whether the larger of the intervals $x - a$ or $x - b$ is smaller than the specified convergence criterion. If convergence has not been achieved, go to step 3.

The MATLAB function *brentopt* implements the Brent search algorithm. The basic syntax is

```
[xopt, fopt, nf] = brentopt(objfun, x1, h, crit)
```

where *objfun* is the objective function, *x1* is the initial point, *h* is the step size, *crit* is the stopping criterion, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *nf* is the number of function evaluations.

```
function [xopt, fopt, nf] = brentopt(objfun, x1, h, crit)
% brentopt.m: minimization(1-dimensional search) by Brent's algorithm
% input:
%   objfun: objective function
%   x1: initial point
%   h: step size
%   crit: convergence criterion
```

```

% output:
%   x: optimum point
%   f: function value at the optimum point
%   hf: number of function evaluations for search
% Sample run:
%   x1 = 0.01; h = 0.2; crit = 1e-6;
%   fun = @(x) exp(x) -3*x + 0.02/x - 0.00004/x;
%   [xopt, fopt, nf] = brentopt(fun, x1, h, crit)

% Initialization
cliffe = 1e40; tau = (sqrt(5) - 1)/2; nf = 1; f1 = objfun(x1);
% Determine initial 3-points
x2 = x1 + h; nf = nf+1; f2 = objfun(x2);
if f2 > f1
    temp = x1; x1 = x2; x2 = temp; temp = f1; f1 = f2; f2 = temp; h = -h;
end
while x1 < cliffe
    h = h/tau; x4 = x2 + h; nf = nf+1; f4 = objfun(x4);
    if f4 > f2, break; end
    f1 = f2; x1 = x2; f2 = f4; x2 = x4;
end
x3 = x4; f3 = f4; a = x1; b = x3; fa = f1; fb = f3; ha = a; hb = b;
% Check whether ha < hb
if (b < a), ha = b; hb = a; end
x = x2; fx = fb; w = x; v = w; ev = 0; fx = f2; fw = fx; fv = fw;
while 1
    hm = (ha + hb)/2;
    % Check interval convergence
    if (abs(x - hm) <= crit - (hb - ha)/2), x2 = x; f2 = fx; return; end
    etemp = 0; p = 0; q = 0;
    if (abs(ev) > crit)
        r = (x - w)*(fx - fv); q = (x - v)*(fx - fw);
        p = (x - v)*q - (x - w)*r; q = 2*(q - r);
        if (q > 0), p = -p;
        else, q = -q; end
        etemp = ev; ev = d; % length of the larger interval
    end
    ind1 = 1;
    if ((q * (ha - x) - p) < 0), ind1 = -1; end
    ind2 = 1;
    if ((q * (hb - x) - p) < 0), ind2 = -1; end
    if ((abs(p) >= abs(q*etemp/2)) | (ind1 == ind2))
        if (x < hm), ev = hb - x;
        else, ev = ha - x; end
        d = (1 - tau)*ev;
    else
        d = p/q; u = x + d;
        if (((u - ha) < crit) | ((hb - u) < crit))
            if (x < hm), d = crit;
            else, d = -crit; end
        end
    end
    if (abs(d) >= crit)
        u = x + d;
    else
        if (d > 0), u = x + crit; else, u = x - crit; end
    end

```

```

nf = nf+1; fu = objfun(u);
% Set a, b, x, u, v, w for the next iteration
if (fu <= fx)
    if (u < x), hb = x; fb = fx;
    else, ha = x; fa = fx; end
    v = w; fv = fw; w = x; fw = fx; x = u; fx = fu;
else
    if (u < x), ha = u; fa = fu;
    else, hb = u; fb = fu; end
    if ((fu <= fw) | (w == x))
        v = w; fv = fw; w = u; fw = fu;
    elseif ((fu <= fv) | (v == x) | (v == w))
        v = u; fv = fu;
    end
end
% Check function convergence
if ((fa - fx) + (fb - fx) < crit), x2 = x; f2 = fx; return; end
xopt = x2; fopt = f2;
end

```

Example 10.3: Brent's Algorithm

Use Brent's search method to find the minimum of $f(x) = \exp(x) - 3x + \frac{0.02}{x} - \frac{0.00004}{x}$. The initial point can be specified as $x_1 = 0.01$. Let the step size be $h = 0.2$ and the stopping criterion be 1×10^{-6} .

Solution

```

>> x1 = 0.01; h = 0.2; crit = 1e-6;
>> fun = @(x) exp(x) - 3*x + 0.02/x - 0.00004/x;
>> [xopt, fopt, nf] = brentopt(fun, x1, h, crit)
xopt =
    1.0572
fopt =
   -0.2744
nf =
      12

```

10.1.4 SHUBERT-PIYAVSKII METHOD²

Consider a simple maximization problem of the form

Maximize $f(x)$
Subject to $a \leq x \leq b$

The function $f(x)$ is assumed to be Lipschitz continuous: there is a constant C whose value is assumed to be known where $|f(x) - f(y)| \leq C|x - y|$ for any $x, y \in [a, b]$. A sequence of points x_0, x_1, x_2, \dots , converging to the global maximum is generated by sequential construction of a sawtooth cover over the function. The Shubert–Piyavskii algorithm is given below:

1. Choose a first sample point x_0 at the midpoint of the interval $[a, b]$ as $x_0 = \frac{1}{2}(a + b)$.
2. Draw a pair of lines with slope C at x_0 : $y(x) = y_0 + C|x - x_0|$. The intersection of this pair of lines with the end points of the interval $[a, b]$ gives an initial sawtooth consisting of two points given by $[(t_1, z_1), (t_2, z_2)] = [(b, y_0 + c/2(b-a)), (a, y_0 + c/2(b-a))]$.

3. Construct a sequence of the sawtooth vertices $[(t_1, z_1), (t_2, z_2), \dots, (t_n, z_n)]$ ($z_1 \leq z_{21} \leq \dots \leq z_n$).
4. Determine the new sawtooth cover $[(t_1, z_1), (t_2, z_2), \dots, (t_{n-1}, z_{n-1}), (t_l, z_l), (t_r, z_r)]$ where $z_l = z_r = \frac{1}{2}(z_n + y_n)$, $t_l = t_n - \frac{1}{2C}(z_n - y_n)$, $t_r = t_n + \frac{1}{2C}(z_n - y_n)$.
5. Stop the procedure when $|z_n - y_n| < \epsilon$.

The MATLAB function *shubertopt* implements the Shubert–Piyavskii algorithm. The basic syntax is

```
[xopt, fopt, nf] = shubertopt(objfun, a, b, C, crit, nfmax)
```

where *objfun* is the objective function, *x1* is the initial point, *a* and *b* are the end points of the initial interval, *C* is the Lipschitz constant, *crit* is the stopping criterion, *nfmax* is the maximum number of function evaluations, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *nf* is the number of function evaluations.

```
function [xopt, fopt, nf] = shubertopt(objfun, a, b, C, crit, nfmax)
% shubertopt.m: optimization by Shubert-Piyavskii algorithm
%                 1-D maximization of Lipschitz functions
% inputs:
%   objfun: objective function
%   a,b: end points of initial interval
%   C: Lipschitz constant
%   crit: tolerance
%   nfmax: maximum number of function evaluations
% outputs:
%   xopt: optimum point
%   fopt: function value at the optimum point
%   nf: number of function evaluations
% Sample run:
%   C = 8; a = -3; b = 8; crit = 1e-6; nfmax = 2000;
%   fun = @(x) -sin(x)-sin(3.5*x);
%   [xopt,fopt,nf] = shubertopt(fun, a, b, C, crit, nfmax)

nf = 0; x0 = (a + b)/2; nf = nf + 1; y0 = objfun(x0); ymax = y0; xmax =
x0; fmax = y0 + C*(b - a)/2;
T(1) = b; Z(1) = y0 + C*(b - a)/2; T(2) = a; Z(2) = y0 + C*(b - a)/2; n = 2;
while ((fmax - ymax) > crit & nf <= nfmax)
    tn = T(n); zn = Z(n); nf = nf + 1; yn = objfun(tn);
    if (yn > ymax), ymax = yn; xmax = tn; end
    zL = (zn + yn)/2; zR = zL;
    tL = tn - (zn - yn)/2/C; tR = tn + (zn - yn)/2/C;
    % Replace T(n) and Z(n) by tl,zl and tr,zr
    ind1 = 0; ind2 = 0;
    if (tL >= a & tL <= b), ind1 = 1; end
    if (tR >= a & tR <= b), ind2 = 1; end
    if (ind1 == 1 & ind2 == 0)
        T(n) = tL; Z(n) = zL;
    elseif (ind1 == 0 & ind2 == 1)
        T(n) = tR; Z(n) = zR;
    elseif (ind1 == 1 & ind2 == 1)
        T(n) = tL; Z(n) = zL; T(n+1) = tR; Z(n+1) = zR; n = n+1;
    end
    [Z, Indx] = sort(Z); T = T(Indx); fmax = Z(n);
end
xopt = xmax; fopt = ymax;
end
```

Example 10.4: Shubert–Piyavskii Algorithm

Use the Shubert–Piyavskii method to find the maximum of $f(x) = -\sin(1.2x) - \sin(3.5x)$. The initial interval can be specified as $[-3, 8]$. Let the Lipschitz constant be $C = 8$, the stopping criterion be 1×10^{-6} , and the maximum number of function evaluations be 2000.

Solution

```

>> C = 8; a = -3; b = 8; crit = 1e-6; nfmax = 2000;
>> fun = @(x) -sin(1.2*x)-sin(3.5*x);
>> [xopt,fopt,nf] = shubertopt(fun, a, b, C, crit, nfmax)
xopt =
    3.2162
fopt =
    1.6239
nf =
    2000

```

10.1.5 STEEPEST DESCENT METHOD

Consider a minimization problem of the form

$$\text{Minimize } f(x)$$

where $x = [x_1, x_2, \dots, x_n]^T$ is a column vector of n real-valued variables. Let x_k be the point at the k th iteration. Let's choose a downhill direction d and a step size $\alpha > 0$ such that the new point $x_k + \alpha d$ yields an improved function value: $f(x_k + \alpha d) < f(x_k)$. To select d , we can use the Taylor expansion on $f(x)$ about x_k :

$$f(x_k + \alpha d) = f(x_k) + \alpha \nabla f(x_k)^T d + O(\alpha^2)$$

where $\nabla f(x)^T = [\partial f(x)/\partial x_1, \partial f(x)/\partial x_2, \dots, \partial f(x)/\partial x_n]$. We can see that δf , the change in f , is given as

$$\delta f = f(x_k + \alpha d) - f(x_k) = \alpha \nabla f(x_k)^T d + O(\alpha^2) \cong \alpha \nabla f(x_k)^T d$$

For $\delta f < 0$ (i.e., reduction in f), d should be a descent direction that satisfies

$$\nabla f(x_k)^T d < 0$$

In the steepest descent method, the direction $d = d_k$ at the k th iteration is chosen as

$$d_k = -\nabla f(x_k)$$

This choice of $d = d_k$ satisfies $\nabla f(x_k)^T d < 0$ since

$$\nabla f(x_k)^T d = \nabla f(x_k)^T d_k = -\nabla f(x_k)^T \nabla f(x_k) = -\|\nabla f(x_k)\|^2 < 0$$

Next we have to determine the step size $\alpha = \alpha_k$ along the direction d_k . As we move along d_k , the objective function f and the design variable depend only on $\alpha = \alpha_k$:

$$f(x_k + \alpha d_k) \equiv f(\alpha), \quad x_k + \alpha d_k \equiv x(\alpha)$$

The directional derivative of the function f along the direction d_k can be expressed as

$$\frac{df(\alpha^*)}{d\alpha} = \nabla f(x_k + \alpha^* d_k)^T d_k$$

We may choose $\alpha = \alpha^*$ so as to minimize $f(\alpha) \equiv f(x_k + \alpha d_k)$. The steepest descent algorithm is presented as follows:

1. Specify a starting point x_0 and stopping criteria ε . Set $k = 0$.
2. Calculate $\nabla f(x_k)$. Stop if $\|\nabla f(x_k)\| \leq \varepsilon$. Otherwise, calculate a normalized direction vector $d_k = -\nabla f(x_k)/\|\nabla f(x_k)\|$.
3. Determine $\alpha_k = \alpha^*$ using an approximate line search method. Update design variable: $x_{k+1} = x_k + \alpha_k d_k$.
4. Calculate the function value at the new point $f(x_{k+1})$. Stop if $|f(x_{k+1}) - f(x_k)| \leq \varepsilon$. Otherwise, set $k = k + 1$, $x_k = x_{k+1}$ and go to step 2.

The MATLAB function *sdopt* implements the steepest descent algorithm. The basic syntax is

```
[xopt, fopt, iter] = sdopt(fun, delfun, x0, alpha0, crit, kmax)
```

where *fun* is the objective function, *delfun* is the gradient of the objective function, *x0* is the initial point vector, *alpha0* is the initial step size, *crit* is the stopping criterion, *kmax* is the maximum number of iterations, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

```
function [xopt, fopt, iter] = sdopt(fun, delfun, x0, alpha0, crit, kmax)
% sdopt.m: steepest descent method
% inputs:
%   fun: objective function
%   delfun: gradient of fun
%   x0: starting point
%   alpha0: initial step size
%   crit: stopping criterion
%   kmax: maximum iterations
% outputs:
%   xopt: optimal point
%   fopt: function value at the optimal point (=f(xopt))
%   iter: number of iterations
% Example:
%   fun = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
%   delfun = @(x) [-400*x(1)*(x(2)-x(1)^2)+2*(x(1)-1); 200*(x(2)-x(1)^2)];
%   x0 = [-1.2 1]; crit = 1e-6; alpha0 = 5; kmax = 1e3;
%   [xopt, fopt, iter] = sdopt(fun, delfun, x0, alpha0, crit, kmax)

nfmax = 10; % maximum function calls during a line search
ni = 2; % indicate poor interval reductions before sectioning
h = alpha0; beta = 0.9; xz = x0; f = fun(x0); x1 = 0; f1 = f; fs = f; k =
0; nc = 0; fold = f;
while (1)
    k = k + 1;
    if k > kmax, disp('Number of maximum iterations exceeded.'), break; end
    x = xz; delf = delfun(x);
    if abs(norm(delf)) <= crit, break; end
    d = -delf'; % steepest descent direction vector
    d = d / norm(d); % row vector
    [xs, fs] = quadappx(fun, xz, x, d, x1, f1, h, nfmax, ni, crit);
    x1 = 0; f1 = fs; xz = xz + xs*d; h = beta * xs; fpr = fs;
    if abs(fpr - fold) < crit
        nc = nc + 1;
        if nc == ni, break; end
    end
end
```

```

else, nc = 0;
end
fold = fpr;
end
iter = k; xopt = x; fopt = fs;
end

function [x2, f2] = quadappx(fun, xz, x, d, x1, f1, h, nfmax, ni, crit)
% Quadratic approximation method for line search
tau = (sqrt(5)-1)/2; x2 = x1 + h; f2 = fun(xz + x2*d);
if f2 < f1
    while (1)
        h = h / tau; x3 = x2 + h; f3 = fun(xz + x3*d);
        if f3 > f2, break;
        else, f1 = f2; x1 = x2; f2 = f3; x2 = x3;
        end
    end
else
    x3 = x2; f3 = f2;
    while (1)
        x2 = (1 - tau) * x1 + tau * x3; f2 = fun(xz + x2*d);
        if f2 <= f1, break;
        else, x3 = x2; f3 = f2;
        end
    end
end
sf = 0.05; % 0 < sf < 0.5
if (x1 >= x2 | x2 >= x3), disp('Incorrect interval.'); return;
elseif (f1 <= f2 | f2 >= f3), disp('Not 3-point pattern.'); return;
end
vs = 0; vc = 0; wc = 0; j = 1;
while j <= nfmax
    sold = abs(x3-x1); fmold = (f1+f2+f3)/3.;
    if vs == 0
        A = (x1-x2)*(x1-x3); B=(x2-x1)*(x2-x3); C=(x3-x1)*(x3-x2);
        x4 = (f1*(x2+x3)/A + f2*(x1+x3)/B + f3*(x1+x2)/C)/(f1/A+f2/B+f3/C)/2;
    else
        if x2 <= (x1+x3)/2, x4 = x2 + (1-tau)*(x3-x2);
        else, x4 = x3 - (1-tau)*(x2-x1);
        end
        vs = 0;
    end
    %safeguard against coincident points
    dxs = sf*min(abs(x2-x1), abs(x3-x2));
    if abs(x4-x1) < dxs, x4 = x1+dxs;
    elseif abs(x4-x3) < dxs, x4 = x3-dxs;
    elseif abs(x4-x2) < dxs
        if x2 > (x1+x3)/2, x4 = x2-dxs;
        else, x4 = x2+dxs;
        end
    end
    f4 = fun(xz + x4*d);
    if (x4 > x2)
        if (f4 >= f2), x3 = x4; f3 = f4;
        else, x1 = x2; f1 = f2; x2 = x4; f2 = f4;
        end
    end
end

```

```

else
    if (f4 >= f2), x1 = x4; f1 = f4;
    else, x3 = x2; f3 = f2; x2 = x4; f2 = f4;
    end
end
snew = abs(x3-x1); fmnew = (f1+f2+f3)/3.;
if abs(x3-x1) <= crit, break; end
if abs(fmnew-fmold) <= crit
    wc = wc + 1;
    if wc == 2, break; end
else, wc = 0;
end
if snew/sold > tau
    vc = vc + 1;
    if vc == ni, vc = 0; vs = 1; end
else, vc = 0; vs = 0;
end
end
end

```

Example 10.5: Steepest Descent Method

Use the steepest descent method to find the minimum of Rosenbrock's function given by

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

The initial point can be specified as $[-1.2, 1]$. Use the initial step size of $\alpha_0 = 5$. The stopping criterion and the maximum number of function evaluations can be set as 1×10^{-6} and 10,000, respectively.

Solution

The gradient of the given function is given by

$$\nabla f(x) = \begin{bmatrix} 400x_1(x_1^2 - x_2) + 2(x_1 - 1) \\ 200(x_2 - x_1^2) \end{bmatrix}$$

```

>> fun = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
>> delfun = @(x) [400*x(1)*(x(1)^2-x(2))+2*(x(1)-1); 200*(x(2)-x(1)^2)];
>> x0 = [-1.2 1]; crit = 1e-6; alpha0 = 5; kmax = 1e4;
>> [xopt,fopt, iter] = sdopt(fun, delfun, x0, alpha0, crit, kmax)
xopt =
    0.9761    0.9526
fopt =
    5.7226e-04
iter =
        2352

```

10.1.6 NEWTON'S METHOD

Newton's method uses second derivatives, which is called the Hessian matrix. The basic idea is to construct a quadratic approximation to the objective function $f(x)$ and minimize the quadratic approximation. At a point x_k , the quadratic approximation can be constructed as

$$s(x) = f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2} (x - x_k)^T \nabla^2 f(x_k) (x - x_k)$$

The minimum of $s(x)$ is found by setting $\nabla s(x) = 0$, which gives

$$[\nabla^2 f(x_k)]d_k = -\nabla f(x_k)$$

where $d_k = x_{k+1} - x_k$. Thus, a new point is given by

$$x_{k+1} = x_k + d_k = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$$

At the new point, we calculate again the gradient and the Hessian matrix and update the point. Newton's method is presented as follows:

1. Choose a starting point x_0 . Set the stopping criterion ϵ and iteration index $k = 0$.
2. Calculate $\nabla f(x_k)$ and $\nabla^2 f(x_k)$. Stop if $\|\nabla f(x_k)\| \leq \epsilon$. Otherwise, find the direction vector d_k .
3. Update the current point by $x_{k+1} = x_k + d_k$.
4. Calculate $f(x_{k+1})$. Stop if $|f(x_{k+1}) - f(x_k)| \leq \epsilon$ is satisfied for two successive iterations. Otherwise, set $k = k + 1$, $x_k = x_{k+1}$ and go to step 2.

The MATLAB function *newtonopt* implements Newton's method. The basic syntax is

```
[xopt, fopt, iter] = newtonopt(fun, x0, crit, kmax)
```

where *fun* is the objective function, *x0* is the initial point vector, *crit* is the stopping criterion, *kmax* is the maximum number of iterations, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations. The Hessian matrix is calculated by the function *jacob*.

```
function [xopt, fopt, iter] = newtonopt(fun, x0, crit, kmax)
% newtonopt.m: minimization by Newton's method
% inputs:
%   fun: objective function
%   x0: starting point
%   crit: stopping criterion
%   kmax: maximum iterations
% outputs:
%   xopt: optimal point
%   fopt: function value at the optimal point (=f(xopt))
%   iter: number of iterations
% Example:
%   fun = @(x) 100*(x(2)-x(1))^2 + (1-x(1))^2;
%   x0 = [-1.2 1]; crit = 1e-6; kmax = 1e3;
%   [xopt, fopt, iter] = newtonopt(fun, x0, crit, kmax)

h = 1e-4; fx = fun(x0); nf = length(fx); nx = length(x0);
xs(1,:) = x0(:)'; % initial row solution vector
for k = 1:kmax
    dx = -jacob(fun, xs(k,:), h)\fx(:); % -[df]^( -1) *fx
    xs(k+1,:) = xs(k,:) + dx'; fx = fun(xs(k+1,:));
    if norm(fx) < crit | norm(dx) < crit, break; end
end
x = xs(k+1,:); xopt = x; fopt = fx; iter = k;
end
```

```

function Hs = jacob(fun, x, h)
% Jacobian of f(x)
hd = 2*h; n = length(x); x = x(:)'; M = eye(n);
for k = 1:n
    Hs(:,k) = (fun(x + M(k,:)*h) - fun(x-M(k,:)*h))'/hd;
end
end

```

Example 10.6: Newton's Method

Use Newton's method to find the minimum of Rosenbrock's function given by

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

The initial point can be specified as $[-1.2, 1]$. The stopping criterion and the maximum number of function evaluations can be set as 1×10^{-6} and 1000, respectively.

Solution

```

>> fun = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
>> x0 = [-1.2 1]; crit = 1e-6; kmax = 1e3;
>> [xopt,fopt, iter] = newtonopt(fun, x0, crit, kmax)
xopt =
    0.9996    0.9993
fopt =
    9.6530e-07
iter =
    646

```

10.1.7 CONJUGATE GRADIENT METHOD

If $f(x)$ is quadratic and is minimized in each search direction, it converges in at most n iterations because its search directions are conjugate. The conjugate gradient method can find the minimum of a quadratic objective function of n variables in n iterations. This method combines current information about the gradient vector with that of gradient vectors from previous iterations to get the new search direction. The new direction is calculated by a linear combination of the current gradient and the previous search direction. This method is also powerful on general non-quadratic functions. Consider the minimization of a quadratic function given by

$$s(x) = \frac{1}{2}x^T Ax + c^T x$$

where

$x = [x_1, x_2, \dots, x_n]^T$ is a column vector of n real-valued variables

A is a positive definite symmetric matrix

c is a constant column vector

The gradient of $s(x)$ at the point x_k can be expressed as

$$\nabla s(x_k) = g_k = Ax_k + c$$

The search direction d is chosen as the steepest descent direction and the first direction d_0 is chosen as $-g_0$. A new point x_{k+1} is determined by minimizing $s(x)$ along the direction d_k as follows:

$$x_{k+1} = x_k + \alpha_k d_k$$

The value of α_k is obtained from the minimization of $f(\alpha) = s(x_k + \alpha d_k)$. From $df(\alpha)/d\alpha = 0$, we have

$$\alpha_k = -\frac{d_k^T g_k}{d_k^T A d_k}, \quad d_k^T g_{k+1} = 0$$

The new direction d_{k+1} is given by

$$d_{k+1} = -g_{k+1} + \beta_k d_k, \quad \beta_k = \frac{g_{k+1}^T (g_{k+1} - g_k)}{g_k^T g_k}$$

The update of d_{k+1} by the above equation is referred to as the Polak–Rebiere algorithm. Considering the fact that $g_{k+1}^T g_k = 0$, β_k can be calculated by

$$\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}$$

which is referred to as the Fletcher–Reeves algorithm.³ Figure 10.3 illustrates the update procedure of d_k for a two-variable quadratic function.

The conjugate gradient method is presented as follows:

1. Choose an initial point x_0 . Set $k = 0$ and calculate $d_0 = -\nabla s(x_0)$.
2. Determine α_k and calculate x_{k+1} .
3. Calculate β_k and the next direction d_{k+1} .
4. Set $k = k + 1$ and go to step 2.

The MATLAB function *cgopt* implements the conjugate gradient algorithm. The basic syntax is

```
[xopt, fopt, iter] = cgopt(fun, delfun, x0, alpha0, crit, kmax)
```

where *fun* is the objective function, *delfun* is the gradient of the objective function, *x0* is the initial point vector, *alpha0* is the initial step size, *crit* is the stopping criterion, *kmax* is the maximum number of iterations, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

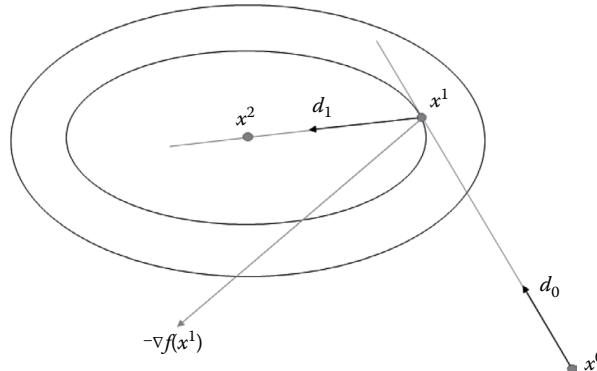


FIGURE 10.3 Conjugate gradient for a two-variable quadratic function. (From Belegundu, A.D. and Chandrupatla, T.R., *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, 2011, p. 108.)

```

function [xopt,fopt, iter] = cgopt(fun, delfun, x0, alpha0, crit, kmax)
% cgopt.m: conjugate gradient method (Fletcher-Reeves algorithm)
% inputs:
%   fun: objective function
%   delfun: gradient of fun
%   x0: starting point
%   alpha0: initial step size (line search)
%   crit: stopping criterion
%   kmax: maximum iterations
% outputs:
%   xopt: optimal point
%   fopt: function value at the optimal point (=f(xopt))
%   iter: number of iterations
% Example:
%   fun = @(x) 100*(x(2)-x(1))^2 + (1-x(1))^2;
%   delfun = @(x) [-400*x(1)*(x(2)-x(1))^2+2*(x(1)-1); 200*(x(2)-x(1)^2)];
%   x0 = [-1.2 1]; crit = 1e-6; alpha0 = 1; kmax = 1e3;
%   [xopt,fopt, iter] = cgopt(fun, delfun, x0, alpha0, crit, kmax)

n = length(x0); % n: number of variables
nfmax = 10; % maximum function calls during a line search
ni = 2; % indicate poor interval reductions before sectioning
h = alpha0; beta = 0.9; xz = x0; f = fun(x0); x1 = 0; f1 = f; fs = f; k = 0;
nc = 0; nd = 0; fold = f;
while (1)
    k = k + 1;
    if k > kmax, disp('Maximum possible iterations exceeded.'), break; end
    nd = nd + 1; x = xz; delf = delfun(x); normd = norm(delf);
    if abs(normd) <= crit, break; end
    if nd == 1, d = -delf'; % row vector
    else, beta = (normd/normd0)^2; d = -delf' + beta*d0;
    end
    d0 = d; d = d/norm(d);
    [xs, fs] = quadappx(fun, xz, x, d, x1, f1, h, nfmax, ni, crit);
    x1 = 0; f1 = fs; xz = xz + xs*d; h = beta * xs; fpr = fs;
    if abs(fpr - fold) < crit
        nc = nc + 1;
        if nc == ni, break; end
    else, nc = 0;
    end
    fold = fpr; normd0 = normd;
    if nd == n+1, nd = 0; end
end
iter = k; xopt = x; fopt = fs;
end

function [x2, f2] = quadappx(fun, xz, x, d, x1, f1, h, nfmax, ni, crit)
% Quadratic approximation method for line search
tau = (sqrt(5)-1)/2;
x2 = x1 + h; f2 = fun(xz + x2*d);
if f2 < f1
    while (1)
        h = h / tau; x3 = x2 + h; f3 = fun(xz + x3*d);

```

```

        if f3 > f2, break;
    else, f1 = f2; x1 = x2; f2 = f3; x2 = x3;
    end
end
else
    x3 = x2; f3 = f2;
    while (1)
        x2 = (1 - tau) * x1 + tau * x3; f2 = fun(xz + x2*d);
        if f2 <= f1, break;
        else, x3 = x2; f3 = f2;
        end
    end
end
sf = 0.05; % 0 < sf < 0.5
if (x1 >= x2 | x2 >= x3)
    disp('Incorrect interval. '); return;
elseif (f1 <= f2 | f2 >= f3)
    disp('Not 3-point pattern. '); return;
end
vs = 0; vc = 0; wc = 0; j = 0;
while j <= nfmax
    sold = abs(x3-x1); fmold = (f1+f2+f3)/3.;
    if vs == 0
        A = (x1-x2)*(x1-x3); B=(x2-x1)*(x2-x3); C=(x3-x1)*(x3-x2);
        x4 = (f1*(x2+x3)/A + f2*(x1+x3)/B + f3*(x1+x2)/C)/(f1/A+f2/
        B+f3/C)/2;
    else
        if x2 <= (x1+x3)/2, x4 = x2 + (1-tau)*(x3-x2);
        else, x4 = x3 - (1-tau)*(x2-x1);
        end
        vs = 0;
    end
    dxs = sf*min(abs(x2-x1), abs(x3-x2));
    if abs(x4-x1) < dxs, x4 = x1+dxs;
    elseif abs(x4-x3) < dxs, x4 = x3-dxs;
    elseif abs(x4-x2) < dxs
        if x2 > (x1+x3)/2, x4 = x2-dxs;
        else, x4 = x2+dxs;
        end
    end
    f4 = fun(xz + x4*d);
    if (x4 > x2)
        if (f4 >= f2), x3 = x4; f3 = f4;
        else, x1 = x2; f1 = f2; x2 = x4; f2 = f4;
        end
    else
        if (f4 >= f2), x1 = x4; f1 = f4;
        else, x3 = x2; f3 = f2; x2 = x4; f2 = f4;
        end
    end
    snew = abs(x3-x1); fmnew = (f1+f2+f3)/3.;
    if abs(x3-x1) <= crit, break; end
    if abs(fmnew-fmold) <= crit
        wc = wc + 1;
        if wc == 2, break; end
    end
end

```

```

else, wc = 0;
end
if snew/sold > tau
    vc = vc + 1;
    if vc == ni, vc = 0; vs = 1; end
else, vc = 0; vs = 0;
end
end
end

```

Example 10.7: Conjugate Gradient Method

Use the conjugate gradient method to find the minimum of Rosenbrock's function given by

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

The initial point can be specified as $[-1.2, 1]$. Use the initial step size of $\alpha_0 = 1$. The stopping criterion and the maximum number of function evaluations can be set as 1×10^{-6} and 10,000, respectively.

Solution

The gradient of the given function is given by

$$\nabla f(x) = \begin{bmatrix} 400x_1(x_1^2 - x_2) + 2(x_1 - 1) \\ 200(x_2 - x_1^2) \end{bmatrix}$$

```

>> fun = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
>> delfun = @(x) [-400*x(1)*(x(2)-x(1)^2)+2*(x(1)-1); 200*(x(2)-x(1)^2)];
>> x0 = [-1.2 1]; crit = 1e-6; alpha0 = 1; kmax = 1e3;
>> [xopt,fopt, iter] = cgopt(fun, delfun, x0, alpha0, crit, kmax)
xopt =
    1.0000    1.0000
fopt =
    1.0380e-10
iter =
    28

```

10.1.8 QUASI-NEWTON METHOD

The search direction of the steepest descent method can be interpreted as being orthogonal to a linear approximation of the objective function at the current point x_k . Newton's method makes use of the 2nd-order approximation of $f(x)$ at x_k to determine a search direction.⁵ The current point x_k can be updated by introducing a step-size parameter α_k as

$$x_{k+1} = x_k + \alpha_k d_k$$

where α_k is obtained from the minimization of $f(\alpha) = f(x_k + \alpha d_k)$. The direction vector d_k should be a descent direction to the function $f(x)$ at $x = x_k$:

$$\nabla f(x_k)^T d_k < 0 \quad \text{or} \quad -\nabla f(x_k)^T [\nabla^2 f(x_k)]^{-1} \nabla f(x_k) < 0$$

The Hessian can be replaced with a symmetric positive definite matrix F_k defined by

$$F_k = \nabla^2 f(x_k) + \gamma I$$

The parameter γ is chosen such that all eigenvalues of F_k are greater than a scalar $\delta > 0$.⁶ The direction vector d_k can be obtained from

$$d_k = -[F_k]^{-1} \nabla f(x_k) = -H_k \nabla f(x_k)$$

where $H_k = [F_k]^{-1}$. Using this direction vector, the update formula can be written as

$$x_{k+1} = x_k + \alpha_k d_k = x_k - \alpha_k [F_k]^{-1} \nabla f(x_k) = x_k - \alpha_k H_k \nabla f(x_k)$$

The basic concept behind the quasi-Newton method is to start with a symmetric positive definite H and update it so that it contains the Hessian information. From the Taylor expansion, we have

$$\nabla f(x_{k+1}) = \nabla f(x_k) + \nabla^2 f(x_k) \delta_k$$

where $\delta_k = x_{k+1} - x_k$. Rearrangement of this equation gives

$$\nabla^2 f(x_k) \delta_k = \gamma_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

The matrix H_{k+1} should satisfy the quasi-Newton condition $H_{k+1} \gamma_k = \delta_k$. The matrix H_{k+1} can be updated according to

$$H_{k+1} = H_k + a u u^T + b v v^T, \quad H_0 = I$$

If we choose $u = \delta_k$ and $v = H_k \gamma_k$, the update formula can be expressed as

$$H_{k+1} = H_k - \frac{H_k \gamma_k \gamma_k^T H_k}{\gamma_k^T H_k \gamma_k} + \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k}$$

which is referred to as the Davidon–Fletcher–Powell (DFP) method.³ Another well-known update formula is given by

$$H_{k+1} = H_k - \frac{\delta_k \gamma_k^T H_k + H_k \gamma_k \delta_k^T}{\delta_k^T \gamma_k} + \left(1 + \frac{\gamma_k^T H_k \gamma_k}{\delta_k^T \gamma_k} \right) \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k}$$

which is referred to as the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method.

The MATLAB function *dfpopt* implements the Davidon–Fletcher–Powell (DFP) method. The basic syntax is

```
[xopt, fopt, iter] = dfpopt(fun, delfun, x0, alpha0, crit, kmax)
```

where *fun* is the objective function, *delfun* is the gradient of the objective function, *x0* is the initial point vector, *alpha0* is the initial step size, *crit* is the stopping criterion, *kmax* is the maximum number of iterations, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

```
function [xopt, fopt, iter] = dfpopt(fun, delfun, x0, alpha0, crit, kmax)
% dfpopt.m: quasi-Newton method (Davidon-Fletcher-Powell (DFP) algorithm)
% inputs:
%   fun: objective function
%   delfun: gradient of fun
%   x0: starting point
```

```

% alpha0: initial step size (line search)
% crit: stopping criterion
% kmax: maximum iterations
% outputs:
% xopt: optimal point
% fopt: function value at the optimal point (=f(xopt))
% iter: number of iterations
% Example:
% fun = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
% delfun = @(x) [-400*x(1)*(x(2)-x(1)^2)+2*(x(1)-1); 200*(x(2)-x(1)^2)];
% x0 = [-1.2 1]; crit = 1e-6; alpha0 = 1; kmax = 1e3;
% [xopt,fopt, iter] = dfpopt(fun, delfun, x0, alpha0, crit, kmax)

n = length(x0); % n: number of variables
nfmax = 10; % maximum function calls during a line search
ni = 2; % indicate poor interval reductions before sectioning
h = alpha0; beta = 0.9; x = x0'; xz = x; f = fun(x); x1 = 0; f1 = f; fs = f;
k = 0; nc = 0; nd = 0; fold = f; H = eye(n);
while (1)
    k = k + 1;
    if k > kmax, disp('Maximum possible iterations exceeded.'), break; end
    x = xz; delf = delfun(x);
    if abs(norm(delf)) <= crit, break; end
    d = -H*delf; d = d/norm(d);
    [xs, fs] = quadappx(fun, xz, x, d, x1, f1, h, nfmax, ni, crit);
    x1 = 0; f1 = fs; xz = xz + xs*d; delf0 = delfun(xz); delf0 = delf0 - delf;
    d0 = H*delf0; Qh = delf0'*d0;
    if abs(Qh) > 1e-15, H = H - d0*d0'/Qh; end
    d0 = xs*d; Pq = d0'*delf0;
    if abs(Pq) > 1e-15, H = H + d0*d0'/Pq; end
    h = beta * xs; fpr = fs;
    if abs(fpr - fold) < crit
        nc = nc + 1;
        if nc == ni, break; end
    else, nc = 0;
    end
    fold = fpr;
    if nd == n+1, nd = 0; H = eye(n); end
end
iter = k; xopt = x; fopt = fs;
end

function [x2, f2] = quadappx(fun, xz, x, d, x1, f1, h, nfmax, ni, crit)
% Quadratic approximation method for line search
tau = (sqrt(5)-1)/2; x2 = x1 + h; f2 = fun(xz + x2*d);
if f2 < f1
    while (1)
        h = h / tau; x3 = x2 + h; f3 = fun(xz + x3*d);
        if f3 > f2, break;
        else, f1 = f2; x1 = x2; f2 = f3; x2 = x3;
        end
    end
else
    x3 = x2; f3 = f2;
    while (1)
        x2 = (1 - tau) * x1 + tau * x3; f2 = fun(xz + x2*d);

```

```

        if f2 <= f1, break;
        else, x3 = x2; f3 = f2;
        end
    end
sf = 0.05; % 0 < sf < 0.5
if (x1 >= x2 | x2 >= x3)
    disp('Incorrect interval. '); return;
elseif (f1 <= f2 | f2 >= f3)
    disp('Not 3-point pattern. '); return;
end
vs = 0; vc = 0; wc = 0; j = 0;
while j <= nfmmax
    sold = abs(x3-x1); fmold = (f1+f2+f3)/3.;
    if vs == 0
        A = (x1-x2)*(x1-x3); B=(x2-x1)*(x2-x3); C=(x3-x1)*(x3-x2);
        x4 = (f1*(x2+x3)/A + f2*(x1+x3)/B + f3*(x1+x2)/C)/(f1/A+f2/
        B+f3/C)/2;
    else
        if x2 <= (x1+x3)/2, x4 = x2 + (1-tau)*(x3-x2);
        else, x4 = x3 - (1-tau)*(x2-x1);
        end
        vs = 0;
    end
    dxs = sf*min(abs(x2-x1), abs(x3-x2));
    if abs(x4-x1) < dxs, x4 = x1+dxs;
    elseif abs(x4-x3) < dxs, x4 = x3-dxs;
    elseif abs(x4-x2) < dxs
        if x2 > (x1+x3)/2, x4 = x2-dxs;
        else, x4 = x2+dxs;
        end
    end
    f4 = fun(xz + x4*d);
    if (x4 > x2)
        if (f4 >= f2), x3 = x4; f3 = f4;
        else, x1 = x2; f1 = f2; x2 = x4; f2 = f4;
        end
    else
        if (f4 >= f2), x1 = x4; f1 = f4;
        else, x3 = x2; f3 = f2; x2 = x4; f2 = f4;
        end
    end
    snew = abs(x3-x1); fmnew = (f1+f2+f3)/3. ;
    if abs(x3-x1) <= crit, break; end
    if abs(fmnew-fmold) <= crit
        wc = wc + 1;
        if wc == 2, break; end
    else, wc = 0;
    end
    if snew/sold > tau
        vc = vc + 1;
        if vc == ni, vc = 0; vs = 1; end
    else, vc = 0; vs = 0;
    end
end
end

```

Example 10.8: Quasi-Newton Method

Wood's function is given by

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 \\ + 10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1)$$

Use the Davidon–Fletcher–Powell (DFP) method to find the minimum of Wood's function. The initial point can be specified as $[-3, -1, -3, -1]$. Use the initial step size of $\alpha_0 = 2$. The stopping criterion and the maximum number of function evaluations can be set as 1×10^{-6} and 1000, respectively.

Solution

The gradient of the given function is given by

$$\nabla f(x) = \begin{bmatrix} 400x_1(x_1^2 - x_2) + 2(x_1 - 1) \\ 200(x_2 - x_1^2) + 20.2(x_2 - 1) + 19.8(x_4 - 1) \\ 360x_3(x_3^2 - x_4) + 2(x_3 - 1) \\ 180(x_4 - x_3^2) + 20.2(x_4 - 1) + 19.8(x_2 - 1) \end{bmatrix}$$

```
>> fun = @(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2+90*(x(4)-x(3)^2)^2+\\
(1-x(3))^2+10.1*((x(2)-1)^2+(x(4)-1)^2)+19.8*(x(2)-1)*(x(4)-1);
>> delfun = @(x) [400*x(1)*(x(1)^2-x(2))+2*(x(1)-1); 200*(x(2)-\\
x(1)^2)+20.2*(x(2)-1)+19.8*(x(4)-1); 360*x(3)*(x(3)^2-x(4))+2*(x(3)-1);\\
180*(x(4)-x(3)^2)+20.2*(x(4)-1)+19.8*(x(2)-1)];
>> x0 = [-3 -1 -3 -1]; crit = 1e-6; alpha0 = 2; kmax = 1e3;
>> [xopt,fopt, iter] = dfpopt(fun, delfun, x0, alpha0, crit, kmax)
xopt =
1.0000
1.0000
1.0000
1.0000
fopt =
1.2396e-12
iter =
39
```

10.2 LINEAR PROGRAMMING

10.2.1 FORMULATION OF LINEAR PROGRAMMING PROBLEM

The general form of the linear programming (LP) problem consists of an objective function to be minimized or maximized and a set of constraints. The general form of LP problem can be stated as

$$\text{Minimize } f(x) = c^T x = c_1 x_1 + c_2 x_2 + \dots + c_n x_n \\ \text{Subject to } Ax = b, x \geq 0$$

where $x = [x_1, x_2, \dots, x_n]^T$ is a column vector of n real-valued variables. The constraints $Ax = b$ can be expressed as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n &= b_m \end{aligned}$$

The constraints can take the form of inequality equations. The inequality constraints can be converted into the equality forms by introducing slack variables. If a constraint appears in the form of \leq (less than or equal to) as

$$a_{j1}x_1 + a_{j2}x_2 + \dots + a_{jn}x_n \leq b_j$$

it can be converted into the equality constraint by adding a nonnegative slack variable x_{n+1} as follows:

$$a_{j1}x_1 + a_{j2}x_2 + \dots + a_{jn}x_n + x_{n+1} = b_j$$

Similarly, if a constraint appears in the form of \geq (greater than or equal to) as

$$a_{j1}x_1 + a_{j2}x_2 + \dots + a_{jn}x_n \geq b_j$$

it can be converted into the equality constraint by subtracting a nonnegative x_{n+1} as

$$a_{j1}x_1 + a_{j2}x_2 + \dots + a_{jn}x_n - x_{n+1} = b_j$$

where x_{n+1} is known as a surplus variable.

A feasible region, \mathcal{F} , is the set of all feasible solutions:

$$\mathcal{F} = \{x \mid Ax = b, x \geq 0\}$$

An optimal solution $x = x^*$ is an element of the feasible region: $x^* \in \mathcal{F}$. The value of the objective function at the optimal point, $f(x^*)$, is called the optimal value. If the solution set of LP contains only one element, the LP problem is said to have the unique optimum. If the optimal value, $f(x^*)$, approaches to positive or negative infinity, the LP is said to have unbounded optimum. A pivot operation used in the solution of LP problems is a sequence of elementary row operations that reduce the coefficients of a specified variable to unity in one of the equations and zero elsewhere.

10.2.2 SIMPLEX METHOD

The simplex method starts with an initial basic feasible solution in canonical form. The solution is improved by finding another basic feasible solution if possible. When a particular basic feasible solution is found and cannot be improved by finding new basic feasible solution, it is said that the optimality is reached. If the minimum of the objective function in the feasible region is finite, then a vertex minimizer exists. Suppose that x_0 is a vertex and that it is not a minimizer. The simplex method generates an adjacent vertex x_1 with $f(x_1) < f(x_0)$. This procedure is continued until a vertex minimizer is reached.

Let A_α be the matrix whose rows are the rows of A that are associated with the constraints that are active at x . The matrix A_α is referred to as the active constraint matrix.

Denote A_{a_k} and the index set J as

$$A_{a_k} = \begin{bmatrix} a_{j1}^T \\ a_{j2}^T \\ \vdots \\ a_{jn}^T \end{bmatrix}, \quad J_{k+1} = [j_1, j_2, \dots, j_{l-1}, i^*, j_{l+1}, \dots, j_n]$$

where i^* is the index that achieves the minimum. Given a vertex x_k , a vertex x_{k+1} is adjacent to x_k if $A_{a_{k+1}}$ differs from A_{a_k} by one row. The simplex method can be summarized as follows:

1. Specify x_0 and form A_{a_0} and J_0 . Set $k = 0$.
2. Solve $A_{a_k}^T \mu_k = c$ for μ_k .
3. Solve $A_{a_k} d_k = e_l$ for d_k .
4. Calculate the residual vector $r_k = Ax_k - b$. If the index set is empty, stop. Otherwise, compute $\alpha_k = \min\left(\frac{r_i}{-a_i^T d_k}\right)$.
5. Set $x_{k+1} = x_k + \alpha_k d_k$. Update $A_{a_{k+1}}$ and J_{k+1} . Set $k = k + 1$ and go to step 2.

10.2.3 Two-Phase Simplex Method

The two-phase simplex method used to solve LP problems consists of phase I and phase II. In phase I, the simplex algorithm is used to find whether the LP problem has a feasible solution. If a feasible solution exists, it provides a basic feasible solution in canonical form to start phase II. Phase II uses the simplex algorithm to find whether the problem has a bounded optimum. If a bounded optimum exists, it finds the basic feasible solution that is optimal. The two-phase simplex method is described in the following steps⁷:

1. Rearrange the given system so that all constant terms b_i are positive or zero by changing the signs on both sides of any of the constraint equations.
2. Rewrite the objection function as $c_1 x_1 + c_2 x_2 + \dots + c_n x_n + (-f) = 0$. Introduce to this system a set of artificial variables (which serves as basic variables in phase I) y_1, y_2, \dots, y_m ($y_i \geq 0$) so that it becomes

$$a_{j1} x_1 + a_{j2} x_2 + \dots + a_{jn} x_n + y_j = b_j \quad (b_j \geq 0, j = 1, \dots, m)$$

3. Phase I. Define w as the sum of the artificial variables: $w = y_1 + y_2 + \dots + y_m$. Use the simplex algorithm to find $x_i \geq 0$ ($i = 1, 2, \dots, n$) and $y_i \geq 0$ ($i = 1, 2, \dots, m$). The canonical form can be rearranged as follows:

$$a_{j1} x_1 + a_{j2} x_2 + \dots + a_{jn} x_n + y_j = b_j \quad (b_j \geq 0, j = 1, \dots, m)$$

$$c_1 x_1 + c_2 x_2 + \dots + c_n x_n + (-f) = 0$$

$$d_1x_1 + d_2x_2 + \dots + d_nx_n + (-w) = -w_0, \quad d_i = -(a_{1i} + \dots + a_{ni}), \quad -w_0 = -(b_1 + \dots + b_m)$$

4. If the minimum of $w > 0$, no feasible solution exists and the procedure is terminated. If the minimum of $w = 0$, start phase II by eliminating the w equation and the columns corresponding to each of the artificial variables from the array.
5. Phase II. Apply the simplex method to the adjusted canonical form at the end of phase I to obtain a solution that optimizes the objective function $f(x)$.

The MATLAB function *simplexlp* implements the two-phase simplex method. The basic syntax is

```
[xopt, fopt] = simplexlp(A, b, c, constr)
```

where A is the coefficient matrix of constraints, b is a column vector denoting the right-hand side of constraints, c is the row vector representing the coefficients of the objective function, $constr$ is a string denoting the type of constraints ('<' or '>' for inequalities and '=' for equalities), $xopt$ is the resultant optimum point, and $fopt$ is the function value at the optimum point.

```
function [xopt, fopt] = simplexlp(A, b, c, constr)
% simplexlp.m: 2-phase LP minimization problem
% Problem: Minimize f(x) = c*x subject to Ax constr b, x >= 0
% inputs:
%   A,b: coefficients of constraint equations (and inequalities)
%   c: coefficients of the objective function
%   constr: type of constraints (example: constr = '==<>=')
% outputs:
%   xopt: optimum point
%   fopt: value of the objective function at x=xopt
% Example: Minimize f(x)=2x1 + 3x2 + 2x3 - x4 + x5
%           subject to 3x1-3x2+4x3+2x4-x5=0, x1+x2+x3+3x4+x5=2
%           A = [3 -3 4 2 -1; 1 1 1 3 1]; b = [0; 2]; c = [2 3 2 -1 1]; constr = '==';
%           [xopt, fopt] = simplexlp(A, b, c, constr)

b = b(:); c = c(:)'; [m, n] = size(A); n1 = n; nleq = 0; neq = 0; ncomv = 0;
if length(c) < n, c = [c zeros(1,n-length(c))]; end
for j = 1:m
    temx = zeros(m,1); temx(j) = 1;
    if(constr(j) == '<') % <=: less than or equal to
        A = [A temx]; nleq = nleq + 1;
    elseif(constr(j) == '>') % >=: greater than or equal to
        A = [A -temx];
    else % =: equality constraints
        neq = neq + 1;
    end
end
lenA = length(A);
if nleq == m
    c = [c zeros(1,lenA-length(c))]; A = [A;c]; A = [A [b;0]];
    [maux, A, z] = compsim(A, n1+1:lenA, 1, 1);
else
    A = [A eye(m) b];
    if m > 1, w = -sum(A(1:m,1:lenA));
    else, w = -A(1,1:lenA);
    end
    c = [c zeros(1,length(A)-length(c))]; A = [A;c]; A = [A; [w zeros(1,m) -sum(b)]];
end
```

```

maux = lenA+1:lenA+m; mv = maux; [maux, A, z] = compsim(A, maux, 2, 1);
nc = lenA + m + 1; x = zeros(nc-1, 1); x(maux) = A(1:m, nc); xm = x(mv);
incomv = intersect(maux, mv);
if (any(xm) ~= 0), disp(sprintf('\n\n Empty feasible region\n')); return
else
    if ~isempty(incomv), ncomv = 1; end
end
A = A(1:m+1,1:nc); A =[A(1:m+1,1:lenA) A(1:m+1,nc)];
[maux, A, z] = compsim(A, maux, 1, 2);
end
if (z == inf | z == -inf), return; end
[m, n] = size(A); x = zeros(n,1); x(maux) = A(1:m-1,n);
x = x(1:n); z = -A(m,n); t = find(A(m,1:n-1) == 0);
if length(t) > m-1, disp('There are infinite solutions'); end
if ncomv == 1, disp('Redundant constraint(s).'); end
xopt = x; fopt = z;
end

function [maux, A, z]= compsim(A, maux, k, ph)
% Main loop of the simplex primal algorithm.
% Bland's rule to prevente cycling is used.
[m, n] = size(A); [mi, col] = Bland(A(m,1:n-1));
while ~isempty(mi) & mi < 0 & abs(mi) > eps
    t = A(1:m-k,col);
    if all(t <= 0)
        z = -inf; disp(sprintf('\n Unbounded optimal solution with z=%s\
n',z)); return
    end
    [row, small] = minrtest(A(1:m-k,n),A(1:m-k,col));
    if ~isempty(row)
        if abs(small) <= 100*eps & k == 1, [s,col] = Bland(A(m,1:n-1)); end
        A(row,:)= A(row,:)/A(row,col); maux(row) = col;
        for i = 1:m
            if i ~= row, A(i,:)= A(i,:)-A(i,col)*A(row,:); end
        end
        [mi, col] = Bland(A(m,1:n-1));
    end
end
z = A(m,n);
end

function [m, j] = Bland(D)
% Apply the Bland's rule to the array D.
% m: first negative number in D, j: index of the entry m.
ind = find(D < 0);
if ~isempty(ind), j = ind(1); m = D(j);
else, m = []; j = [];
end
end

function [row, mi] = minrtest(a, b)
% Minimum ratio test on vector a and vector b.
% row: index of the pivot row, mi: value of the minimum ratio.
m = length(a); c = 1:m; a = a(:); b = b(:); l = c(b > 0);
[mi, row] = min(a(l)./b(l)); row = l(row);
end

```

Example 10.9: Two-Phase Simplex Method (1)

Find the solution of the following LP using the two-phase method:

Minimize $f(x) = 2x_1 + x_2$
 Subject to $2x_1 + 5x_2 \geq 20$, $x_1 + x_2 \geq 6$, $3x_1 + x_2 \geq 9$, $x_1 \geq 0$, $x_2 \geq 0$

Solution

$$A = \begin{bmatrix} 2 & 5 \\ 1 & 1 \\ 3 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 20 \\ 6 \\ 9 \end{bmatrix}, \quad c = [2 \ 1]$$

Since all the constraints are inequality (\geq ; greater than or equal to) equations, constr can be set as ' $> > >$ '.

```
>> A = [2 5; 1 1; 3 1]; b = [20; 6; 9]; c = [2 1]; constr = '> > >';  

>> [xopt, fopt] = simplexlp(A, b, c, constr)  

xopt =  

    1.5000  

    4.5000  

fopt =  

    7.5000
```

We can see that the optimum point is $x_1 = 1.5$, $x_2 = 4.5$ and the optimal value is $f(x_{opt}) = 7.5$.

Example 10.10: Two-Phase Simplex Method (2)⁸

In the platform support system shown in Figure 10.4, cable 1 can support 120 lb, cable 2 can support 160 lb, and cables 3 and 4 can support 100 lb each. A weight of x acting at a from the left support and b from the right support causes reactions of $bx/(a + b)$ and $ax/(a + b)$, respectively. Determine the maximum load that the system can support.

Solution

The problem can be formulated as

Maximize $f(x) = x_1 + x_2$ or minimize $f(x) = -x_1 - x_2$
 Subject to $x_2 \leq 200$, $4x_1 + 3x_2 \leq 1280$, $4x_1 + x_2 \leq 960$, $x_1 \geq 0$, $x_2 \geq 0$

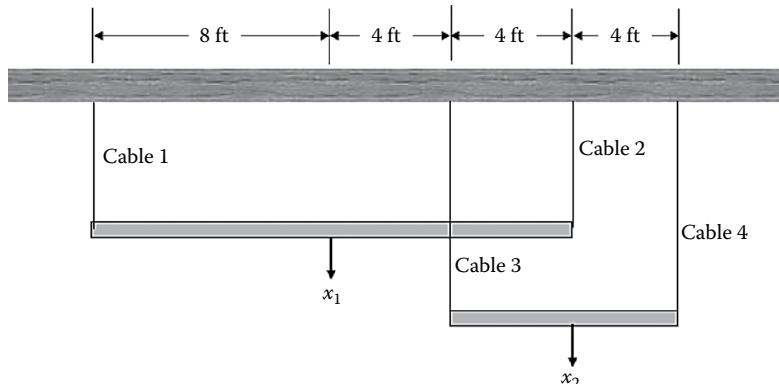


FIGURE 10.4 Platform support system.

```

>> A = [0 1; 4 3; 4 1]; b = [200; 1280; 960]; c = [-1 -1]; constr = '< < < ';
>> [xopt, fopt] = simplexlp(A, b, c, constr)
xopt =
    170
    200
fopt =
   -370

```

10.2.4 INTERIOR POINT METHOD

Consider a linear programming problem rearranged in the standard form as

$$\begin{aligned} & \text{Maximize } f(x) = c^T x \\ & \text{Subject to } Ax = b, x \geq 0 \end{aligned}$$

where $x = [x_1, x_2, \dots, x_n]^T$ is a column vector of n real-valued variables. In the interior point method, we start from a strictly interior point x^0 such that all the elements of x^0 are positive. From this interior point, we proceed in some direction to increase the objective function for maximization. This direction is obtained from the coefficient vector c and the coefficient matrix of constraints A . We use a scaling scheme and then a projection idea to find a reasonable direction. Various scaling schemes may be used, and Karmarkar uses a nonlinear scaling scheme and applies it to an equivalent problem.⁹ The interior point algorithm transforms the linear programming problem to a more convenient form and then searches through the interior of the feasible region using a direction of search toward its surface. The interior point method is described in the following steps¹⁰:

1. Let the number of variables be n and set $a(i, n+1) = b(i) - \sum_j a(i, j)$ and $c(n+1) = 1 \times 10^4$. Specify the initial point $x^0 = [1, 1, \dots, 1]$. Set $k = 0$.
2. Set $D^k = \text{diag}(x^k)$ and calculate an improved point using $x^{k+1} = x^k - \frac{s(D^k)^2(c - A^T \lambda^k)}{\text{norm}(D^k(c - A^T \lambda^k))}$ where $\lambda^k = (A(D^k)^2 A^T)^{-1} A(D^k)^2 c$ and the step s is chosen such that

$$s = \min \left\{ \frac{\text{norm}(D^k(c - A^T \lambda^k))}{x_j^k(c_j - A_j^T \lambda^k)} \right\} - \alpha \quad \text{where } A_j \text{ is the } j\text{th column of the matrix } A.$$

3. Stop if the primal and dual values of the objective functions are approximately equal. Else set $k = k + 1$ and go to step 2.

The MATLAB function *barnslp* implements the interior point method by Barnes' algorithm. The basic syntax is

```
[xopt, fopt, basic] = barnslp(A, b, c, tol)
```

where A is the coefficient matrix of constraints, b is a column vector denoting the right-hand side of constraints, c is the row vector representing the coefficients of the objective function, tol is the stopping criterion, $xopt$ is the resultant optimum point, $fopt$ is the function value at the optimum point, and $basic$ is the list of basic variables.

```

function [xopt, fopt, basic] = barnslp(A, b, c, tol)
% barnslp.m: solution of LP by using Barnes' interior point method
% Problem: Minimize c'x subject to Ax = b (assumed to be non-degenerate)

```

```

% Inputs:
%   A,b: coefficients of constraint equations
%   c: coefficients of the objective function
% Outputs:
%   xopt: solution vector
%   fopt: function value at xsol
%   basic: list of basic variables
% Example: minimize f(x)=2x1 + 3x2 + 2x2 - x4 + x5
%   subject to 3x1-3x2+4x3+2x4-x5=0, x1+x2+x3+3x4+x5=2
%   A = [3 -3 4 2 -1;1 1 1 3 1]; b = [0; 2]; c = [2 3 2 -1 1]; tol = 1e-6;
%   [xopt,fopt, basic] = barns1p(A,b,c,tol)

% Initialization
x2 = [ ]; x = [ ]; [m n] = size(A); ctemp = zeros(1,n); clen = length(c);
for j = 1:clen, ctemp(j) = c(j); end
c = ctemp; aplus1 = b - sum(A(1:m,:))'; cplus1 = 1000000;
A = [A aplus1]; c = [c cplus1]; B = [ ]; n = n+1; x0 = ones(1,n)'; x = x0;
alpha = 0.0001; lambda = zeros(1,m)'; iter = 0;
% Main step
while abs(c*x - lambda'*b) > tol
    x2 = x.*x; D = diag(x); D2 = diag(x2); AD2 = A*D2;
    lambda = (AD2*A')\ (AD2*c'); dualres = c' - A'*lambda; normres =
        norm(D*dualres);
    for i = 1:n
        if dualres(i)>0, ratio(i) = normres/(x(i)*(c(i)-A(:,i)/*lambda));
        else, ratio(i) = inf;
        end
    end
    R = min(ratio) - alpha; x1 = x - R*D2*dualres/normres; x = x1;
    basiscount = 0;
    B = [ ]; basic = [ ]; cb = [ ];
    for k = 1:n
        if x(k)>tol, basiscount = basiscount+1; basic = [basic k]; end
    end
    % Non-degenerate problem
    if basiscount == m
        for k = basic, B = [B A(:,k)]; cb = [cb c(k)]; end
        primalsol = b'/B'; xopt = primalsol; break;
    end
    iter = iter + 1;
end
xopt = x(basic); fopt = c*x;
end

```

Example 10.11: Interior Point Method

Find the solution of the following LP using the interior point method:

$$\begin{aligned}
 & \text{Minimize } f(x) = -2x_1 - x_2 - 4x_3 \\
 & \text{Subject to } x_1 + x_2 + x_3 + x_4 = 7, x_1 + 2x_2 + 3x_3 + x_5 = 12, x_i \geq 0 \ (i = 1, \dots, 5)
 \end{aligned}$$

Solution

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 2 & 3 & 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 7 \\ 12 \end{bmatrix}, \quad c = \begin{bmatrix} -2 & -1 & -4 \end{bmatrix}$$

```

>> A = [1 1 1 1 0; 1 2 3 0 1]; b = [7; 12]; c = [-2 -1 -4]; tol = 1e-6;
>> [xopt, fopt, basic] = barnslp(A,b,c,tol)
xopt =
    4.5000
    2.5000
fopt =
    -19.0000
basic =
    1         3

```

10.3 CONSTRAINED OPTIMIZATION

10.3.1 ROSEN'S GRADIENT PROJECTION METHOD

Consider a simple minimization problem of the form

Minimize $f(x)$

Subject to $a^i x - b_i \leq 0$ ($i = 1, \dots, m$), $a^i x - b_i = 0$ ($i = m + 1, \dots, m + l$)

where $x = [x_1, x_2, \dots, x_n]^T$ is a column vector of n real-valued variables. Now, suppose that x^k is the current design point. It is required that x^k satisfies the above constraints. We have to determine a direction vector d followed by a step length along this direction, which will give us a new and improved design point. Let q be the number of active constraints. We can introduce the tangent plane that can be described by $n - q$ independent parameters. Define a matrix $B \in \mathbb{R}^{q \times n}$ that consists of rows of the gradient vectors of the active constraints:

$$B = \begin{bmatrix} a^{1T} & | & a^{2T} & | & \dots & | & a^q \end{bmatrix}^T$$

We find a direction vector d that satisfies $By = 0$ and that makes $|\nabla f(x^k) - d|$ a minimum. The direction d can be determined as the solution of the following minimization problem:

Minimize $h(d) = (-\nabla f(x^k) - d)^T (-\nabla f(x^k) - d)$

Subject to $Bd = 0$

We can define the Lagrangian $L(d) = (\nabla f(x^k) + d)^T (\nabla f(x^k) + d) + \beta^T Bd$ and apply the optimality condition as follows:

$$\frac{\partial L^T}{\partial d} = \left(\nabla f(x^k)^T + d \right) + B^T \beta = 0$$

Using the constraint equation $Bd = 0$, we have

$$BB^T \beta = -B \nabla f(x^k)$$

Solving this equation, we have

$$\beta = -\left(BB^T \right)^{-1} B \nabla f(x^k)$$

Thus, the direction vector d is given by

$$d = -\nabla f(x^k) - B^T \beta = -\nabla f(x^k) + B^T \left(BB^T \right)^{-1} B \nabla f(x^k) = W \left(-\nabla f(x^k) \right)$$

where

$$W = I - B^T (BB^T)^{-1} B$$

Rosen's gradient projection method for linear constraints is described in the following steps¹¹:

1. Choose a feasible starting point x^0 .
2. Determine the active set and construct the matrix B .
3. Calculate β and d .
4. If $d \neq 0$, determine α_k and update the current design point by $x^{k+1} = x^k + \alpha_k d$ and go to step 2.
5. If $\beta_j \geq 0$ for all j corresponding to active inequalities, stop the procedure. Otherwise, delete the row from B corresponding to the most negative component of β and go to step 3.

The MATLAB function *rosencopt* implements Rosen's gradient projection method. The basic syntax is

```
[xopt, fopt, iter] = rosencopt(fun, delfun, x0, A, b, ne, m, crit)
```

where *fun* is the objective function, *delfun* is the gradient of the objective function, *x0* is the initial point vector, *A* is the coefficient matrix of constraints, *b* is a column vector denoting the right-hand side of constraints, *ne* is the number of equality constraints, *m* is the number of inequality constraints, *crit* is the stopping criterion, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

```
function [xopt, fopt, iter] = rosencopt(fun, delfun, x0, A, b, ne, m, crit)
% rosencopt.m: Rosen's gradient projection method
% Problem type:
%   Minimize f(x)
%   subject to Aj*x = b (j=1,...,ne), Aj*x <= b (j=ne+1,...,m)
%   Starting point should satisfy all constraints.
% inputs:
%   fun: objective function
%   delfun: gradient of fun
%   x0: starting point (must satisfy all constraints)
%   A: constraints coefficient matrix
%   b: right-hand side of constraints (column vector)
%   ne: number of equality constraints
%   m: number of inequality constraints
%   crit: stopping criterion
% outputs:
%   xopt: optimal point
%   fopt: function value at the optimal point (=f(xopt))
%   iter: number of iterations
% Example:
%   fun = @(x) -x(1)*x(2)*x(3);
%   delfun = @(x) [-x(2)*x(3); -x(1)*x(3); -x(1)*x(2)];
%   x0 = 10*[1 1 1]; crit = 1e-4; ne = 0; m = 8;
%   A = [-1 0 0; 0 -1 0; 0 0 -1; 1 0 0; 0 1 0; 0 0 1; -1 -2 -2; 1 2 2];
%   b = [0 0 0 42 42 42 0 72]';
%   [xopt, fopt, iter] = rosencopt(fun, delfun, x0, A, b, ne, m, crit, kmax)

x = x0; n = length(x0); % n: number of variables
f = fun(x); iter = 0;
```

```

while (1)
    iter = iter + 1; [nc, nca] = actcont(n,m,ne,crit,x,A,b); % active
    constraints
    [dv,d,Bm] = dirvec(delfun,n,nc,ne,crit,x,nca,A); % direction (row) vector
    if (abs(dv) < crit), break; end
    alphak = maxstep(delfun,n,m,nc,x,x0,d,A,b,nca); % maximum step size
    x = x0 + alphak*d; fp = delfun(x)'*d';
    if (fp > 0), x = bisec(delfun,n,alphak,x0,d); end
    f = fun(x); x0 = x;
end
xopt = x; fopt = f;
end

function [nc, nca] = actcont(n,m,ne,crit,x,A,b)
g = A*x' - b;
for k = 1:m, nca(k) = k; end
nc = ne;
for j = ne+1:m
    if (abs(g(j)) < crit)
        nc = nc + 1; ntemp = nca(j); nca(j) = nca(nc); nca(nc) = ntemp;
    end
end
end

function x = bisec(delfun,n,alphak,x0,d)
mcrit = 1e-6; a1 = 0; a2 = alphak; aw = a2 - a1;
while (a2 - a1) > mcrit*aw
    am = (a1 + a2)/2; x = x0 + am*d; fp = delfun(x)'*d';
    if (fp < 0), a1 = am;
    elseif (fp > 0), a2 = am;
    else, break;
    end
end
x = x0 + a1*d;
end

function [dn,d,Bm] = dirvec(delfun,n,nc,ne,crit,x,nca,A)
df = delfun(x)'; % row vector
while (1)
    if (nc == 0)
        d = -df; dn = norm(d);
        if (dn < crit), break; end
        d = d/dn; Bm(1) = 0; return;
    else
        for j = 1: nc
            ic = nca(j);
            for jk = 1: nc
                jc = nca(jk); Am(j,jk) = 0.;
                for k = 1: n, Am(j,jk) = Am(j,jk) + A(ic,k) * A(jc,k); end
            end
        end
        for j = 1: nc
            ic = nca(j); Bm(j) = 0.;
            for k = 1: n, Bm(j) = Bm(j) - A(ic,k) * df(k); end
        end
    end
end

```

```

Am = Am(1:nc,1:nc); Bm = Bm(1:nc)'; Bm = inv(Am)*Bm;
for j = 1: n
    d(j) = -df(j);
    for k = 1: nc, kn = nca(k); d(j) = d(j) - A(kn,j) * Bm(k); end
end
dn = norm(d);
if (nc == ne & dn <= crit), break; end
if (dn <= crit)
    Bmin = Bm(ne+1); imin = ne + 1;
    for j = ne + 1: nc
        if (Bmin > Bm(j)), Bmin = Bm(j); imin = j; end
    end
    if (Bmin >= 0.), break;
    else, ntemp = nca(imin); nca(imin) = nca(nc); nca(nc) = ntemp; nc
        = nc - 1;
    end
else, d = d/dn; break;
end
end
function alphak = maxstep(delfun,n,m,nc,x,x0,d,A,b,nca)
nq = 0;
for k = nc + 1: m
    cq = nca(k); c = -b(cq); aq = 0.;
    for j = 1:n, c = c + A(cq,j)*x(j); aq = aq + A(cq,j)*d(j); end
    if (aq ~= 0)
        am = -c/aq;
        if (am > 0.)
            nq = nq + 1;
            if (nq == 1), alphak = am;
            else
                if (alphak > am), alphak = am; end
            end
        end
    end
end
if (nq == 0)
    alphak = 1;
    while (2)
        x = x0 + alphak*d; fp = delfun(x) '*d';
        if (fp > 0.), break; end
        alphak = 2*alphak;
    end
end
end

```

Example 10.12: Rosen's Gradient Projection Method

Find the solution of the following constrained minimization problem using Rosen's gradient projection method:

$$\begin{aligned}
 & \text{Minimize } f(x) = 0.02x_1^2 + 1.2x_2^2 - 80 \\
 & \text{Subject to } 3 - x_1 \leq 0, 12 - 10x_1 + x_2 \leq 0, -40 \leq x_1, x_2 \leq 40
 \end{aligned}$$

As a starting point, use $x^0 = [4, 5]$. Note that this point satisfies all constraints.

Solution

The constraints can be rearranged as follows:

$$-x_1 \leq -3, \quad 10x_1 + x_2 \leq -12, \quad x_1 \leq 40, \quad x_2 \leq 40, \quad x_1 \leq 40, \quad x_2 \leq 40$$

From these inequalities, we have

$$A = \begin{bmatrix} -1 & 0 \\ -10 & 1 \\ -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} -3 \\ -12 \\ 40 \\ 40 \\ 40 \\ 40 \end{bmatrix}$$

```
>> fun = @(x) 0.02*x(1)^2+1.2*x(2)^2-80;
>> delfun = @(x) [0.04*x(1); 2.4*x(2)];
>> ne = 0; m = 6; crit = 1e-4; x0 = [4 5];
>> A = [-1 0;-10 1;-1 0;0 -1;1 0;0 1]; b = [-3 -12 40 40 40 40]';
>> [xopt,fopt,iter] = rosencopt(fun,delfun,x0,A,b,ne,m,crit)
xopt =
    3.0000    0.0000
fopt =
    -79.8200
iter =
    4
```

10.3.2 ZOUTENDIJK'S FEASIBLE DIRECTION METHOD

Consider a minimization problem with nonlinear constraints of the form

$$\begin{aligned} & \text{Minimize } f(x) \\ & \text{Subject to } g_i(x) \leq 0 \ (i = 1, \dots, m) \end{aligned}$$

where $x = [x_1, x_2, \dots, x_n]^T$ is a column vector of n real-valued variables. We first define the active set H as

$$H = \left\{ g_j(x^k) + \varepsilon \geq 0, \quad i = 1, \dots, m \right\}$$

Next we introduce an artificial variable α , which is given by

$$\alpha = \max \left\{ \nabla f(x^k)^T d, \nabla g_j(x^k)^T d, \quad j \in H \right\}$$

To obtain a descent feasible direction, we have to reduce α until it becomes a negative number. To do this, we can formulate the following subproblem:

$$\begin{aligned} & \text{Minimize } \alpha \\ & \text{Subject to } \nabla f(x^k)^T d \leq \alpha, \quad \nabla g_j(x^k)^T d \leq \alpha, \quad j \in H, \quad -1 \leq d_i \leq 1 \quad (i = 1, \dots, n) \end{aligned}$$

The step-size problem is a constrained one-dimensional search and can be described as follows:

$$\begin{aligned} \text{Minimize } f(\alpha) &= f(x^k + \alpha d) \\ \text{Subject to } g_j(\alpha) &= g_j(x^k + \alpha d) \leq 0 \quad (i = 1, \dots, m) \end{aligned}$$

This problem can be rewritten as

$$\begin{aligned} \text{Minimize } -\beta &= \alpha \\ \text{Subject to } \nabla f(x^k)^T s + \beta &\leq c_0, \quad \nabla g_j(x^k)^T s + \theta_j \beta \leq c_j \quad (j \in H), \quad 0 \leq s_i \leq 2 \quad (i = 1, \dots, n), \quad \beta \geq 0 \end{aligned}$$

where $c_0 = \sum_{i=1}^n \partial f / \partial x_i$, $c_j = \sum_{i=1}^n \partial g_j / \partial x_i$. Zoutendijk's feasible direction method can be summarized as follows¹²:

1. Choose a feasible starting point x^0 that satisfies the constraints.
2. Determine the set H .
3. Solve the minimization problem (minimize $-\beta = \alpha$) to find β^* and d^* .
4. If $\beta^* = 0$, stop the procedure.
5. Otherwise, find the optimum step size α_k and update the current point as $x_{k+1} = x_k + \alpha_k d^*$ and go to step 2.

The MATLAB function *zoutopt* implements Zoutendijk's feasible direction method. The basic syntax is

```
[xopt, fopt, iter] = zoutopt(funz, delf, delg, x0, xl, xu, nc, ncs, crit, kmax)
```

where *funz* is the objective function and constraints, *delf* is the gradient of the objective function, *delg* is the gradient of constraints, *x0* is the starting point vector, *xl* and *xu* are the lower and upper bounds on *x*, *nc* is the number of active constraints, *ncs* is the number of constraints, *crit* is the stopping criterion, *kmax* is the number of maximum possible iterations, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

```
function [xopt, fopt, iter] = zoutopt(funz, delf, delg, x0, xl, xu, nc, ncs, crit, kmax)
% zoutopt.m: minimization by Zoutendijk's feasible direction method
% Problem type: min. f(x)
% subject to gj(x) <= 0 (j=1,...,nca), xl <= x <= xu
% Inputs:
% funz: objective function and constraints
% delf: gradient of f
% delg: gradient of g
% x0: starting point (must satisfy all constraints)
% xl,xu: lower and upper limit of x
% nc: number of active constraints
% ncs: number of constraints
% crit: stopping criterion
% kmax: maximum possible iterations
% Outputs:
% xopt: optimal point
% fopt: function value at the optimal point (=f(xopt))
% iter: number of iterations
% Example:
% delf = @(x) [-1; -2]; delg = @(x) [2*x(1); 12*x(2)];
% x0=[1 0]; xl=[0 0]; xu=[10 10]; nc=0; ncs=1; crit=1e-4; kmax=1e3;
% [xopt, fopt, iter] = zoutopt(funz, delf, delg, x0, xl, xu, nc, ncs, crit, kmax)
```

```

mcrit = 2e-3; gcrit = 1e-4; maxac = 20; maxac = 20; maxs = 30;
nv = length(x0); x = x0; fg = funz(x); f = fg(1); g = fg(2);
ic = 0; fold = f; iter = 0; f0 = f;
while (1)
    iter = iter + 1;
    [nc, na] = actcont(ncs, crit, g); % active constraints
    if (iter > kmax), disp('Maximim possible iterations exceeded.');
        break; end
    [d, dn, beta] = dirvec(delf,delg,crit,nv,x,nc,xl,xu,mcrit); %
        direction vector
    if (abs(dn) < crit | abs(beta) < crit), break; end
    d = d/dn;
    alpha = linsr(funz,delf,nv,ncs,x,nc,na,xl,xu,d,x0,maxs,gcrit,crit); %
        calculate step size
    x = x0 + alpha*d; fg = funz(x); f = fg(1); x0 = x;
    if (abs(f - fold) < crit)
        ic = ic + 1;
        if (ic == 2), break; end
    else, ic = 0;
    end
    fold = f;
end
fg = funz(x); f = fg(1); g = fg(2); [nc,na] = actcont(ncs, crit, g); xopt =
    x; fopt = f;
end

function [nc, na] = actcont(ncs, mcrit, g)
for i = 1:ncs, na(i) = i; end
nc = 0;
for k = 1:ncs
    if ( g(k) > -mcrit), nc = nc + 1; ntemp = na(k); na(k) = na(nc);
        na(nc) = ntemp; end
end
end

function [d, dn, beta] = dirvec(delf,delg,crit,nv,x,nc,xl,xu,mcrit)
df = delf(x)'; % row vector
if (nc > 0), A = delg(x)'; end
df = df/norm(df); % normalization
if (nc > 0)
    for j = 1:nc, A(j,:) = A(j,:)/sqrt(A(j,:)*A(j,:)'); end
end
% active bounds
for k = 1:nv
    if (xl(k) - x(k) + mcrit >= 0), nc = nc + 1; A(nc,1:nv) = 0; A(nc,k) =
        -1; end
end
for k = 1:nv
    if (x(k) - xu(k) + mcrit >= 0), nc = nc + 1; A(nc,1:nv) = 0; A(nc,k) =
        1; end
end
if (nc == 0), beta = 1; d = -df; dn = norm(d); return; end
[beta, d] = simpx(nc,nv,df,A); dn = sqrt(d*d'); % feasible direction
end

function alpha = linsr(funz,delf,nv,ncs,x,nc,na,xl,xu,d,x0,maxs,gcrit,crit)
% Determine maximum step size using line search method

```

```

nlarge = 1e40; c = max(abs(xu - xl));
for k = 1:nv
    if (abs(d(k))*nlarge > c)
        if (d(k) < 0)
            cn = (xl(k) - x(k)) / d(k);
            if (cn < nlarge), nlarge = cn; end
        else
            cn = (xu(k) - x(k)) / d(k);
            if (cn < nlarge), nlarge = cn; end
        end
    end
abet = nlarge; x = x0 + abet * d; fg = funz(x); f = fg(1); g = fg(2);
gmax = max(g); inda = 1;
if (gmax <= 0), inda = 0; end
if (inda == 0), amax = abet;
else, xl = 0; [xm, fm] = nears(funz,xl,abet,x,d,x0,maxs,crit); amax = xm;
end
x = x0 + amax*d; df = delf(x)'; sdr = df*d';
if (sdr <= 0), alpha = amax; return; end
a1 = 0; a2 = amax; adif = a2 - a1;
while ((a2 - a1) > crit * adif)
    am = (a1 + a2)/2; x = x0 + am*d; df = delf(x)'; sdr = df*d';
    if (sdr == 0), break; end
    if (sdr < 0), a1 = am;
    elseif (sdr > 0), a2 = am;
    end
end
alpha = a1;
end

function [xm, fm] = nears(funz,xa,xb,x,d,x0,maxs,crit)
miter = 0;
while (1)
    xm = (xa + xb)/2; miter = miter + 1;
    if (miter > maxs), xm = xa; fm = fa; return; end
    x = x0 + xm*d; fg = funz(x); f = fg(1); g = fg(2); gmax = max(g); fm = f;
    if (gmax <= 0 & gmax >= -crit), return; end
    if (gmax < 0), xa = xm; fa = fm;
    else, xb = xm; end
end
end

function [beta, d] = simpex(nc,nv,df,A)
% Find search direction using the linear programming (simplex) method
Bm = 1e2; nrow = nc + nv + 2; nm = nc + nv + 1; Bm(1:nrow) = 0; Bm(1) =
sum(df);
for j = 1:nc
    Bm(j+1) = 0;
    for k = 1: nv, Bm(j+1) = Bm(j+1) + A(j,k); end
end
for k = nc + 2:nrow - 1, Bm(k) = 2; end
for k = 1: nm, Bs(k) = nv + k + 1; end
ncol = nv + nm + 1;
for k = 1:nm
    if (Bm(k) < 0), ncol = ncol + 1; Bs(k) = -ncol; end
end

```

```

Am(1:nrow,1:ncol) = 0; Am(1,1:nv) = df; Am(1,nv+1) = 1;
for k = 1:nc
    for j = 1: nv, Am(k+1,j) = A(k,j); end
    Am(k+1,nv+1) = 1;
end
mi = 0;
for k = nc+2: nrow-1, mi = mi + 1; Am(k,mi) = 1; end
Am(nrow,nv+1) = -1;
for k = 1: nm, Am(k,nv+k+1) = 1; end
nt = nv + nm + 1;
for k = 1: nm
    if (Bm(k) < 0)
        nt = nt + 1; Bm(k) = -Bm(k);
        for j = 1:ncol, Am(k,j) = -Am(k,j); end
        Am(k,nt) = 1; Am(nrow,nt) = Bg;
    end
end
for k = 1:nm
    if (Bs(k) < 0)
        Bs(k) = -Bs(k);
        for j = 1: ncol, Am(nrow,j) = Am(nrow,j) - Bg*Am(k,j); end
        Bm(nrow) = Bm(nrow) - Bg*Bm(k);
    end
end
while (1)
    nf0 = 0;
    for k = 1:ncol
        if (Am(nrow, k) < 0), nf0 = 1; break; end
    end
    if (nf0 == 0), break; end
    c = Bg;
    for j = 1:ncol
        if (Am(nrow, j) < c), c = Am(nrow, j); iv = j; end
    end
    ik = 0; jk = 0;
    for k = 1:nrow - 1
        if (Am(k,iv) > 0)
            jk = jk + 1; c1 = Bm(k)/(Am(k,iv) + 1e-10);
            if (jk == 1), c = c1; jp = k;
            else, if (c1 < c), c = c1; jp = k; end
            end
            ik = 1;
        end
    end
    Bs(jp) = iv;
    if (ik == 0), disp('Unbounded objective function.'); break; end
    c1 = 1/Am(jp,iv); Bm(jp) = c1*Bm(jp);
    for j = 1:ncol, Am(jp,j) = c1*Am(jp,j); end
    for k = 1: nrow
        if (k ~= jp)
            c2 = Am(k,iv);
            for j = 1: ncol, Am(k,j) = Am(k,j) - c2*Am(jp,j); end
            Bm(k) = Bm(k) - c2*Bm(jp);
        end
    end
end

```

```

d(1:nv) = -1;
for k = 1: nm
    for j = 1: nv
        if (j == Bs(k)), d(j) = Bm(k) - 1; end
    end
end
beta = Bm(nrow);
end

```

Example 10.13: Zoutendijk's Feasible Direction Method

Find the solution of the following constrained minimization problem using Zoutendijk's feasible direction method:

$$\begin{aligned}
 & \text{Minimize } f(x) = -(1.2x_1 + 3x_2) \\
 & \text{Subject to } g(x) = x_1^2 + 6x_2^2 - 1 \leq 0, \quad 0 \leq x_1, \quad x_2 \leq 10
 \end{aligned}$$

As a starting point, use $x^0 = [1, 0]$. There are no active constraints ($nc = 0$). Note that this point satisfies all constraints.

Solution

The objective function and the constraint can be defined as a function as follows:

```

function fg = funz(x)
fg(1) = -(1.2*x(1) + 3*x(2));
fg(2) = x(1)^2 + 6*x(2)^2 - 1;
end

```

The gradients of $f(x)$ and $g(x)$ are given by

$$\frac{\partial f}{\partial x_1} = -1.2, \quad \frac{\partial f}{\partial x_2} = -3, \quad \frac{\partial g}{\partial x_1} = 2x_1, \quad \frac{\partial g}{\partial x_2} = 12x_2$$

The lower and upper bounds on x can be written as $x^l = [0 \ 0]$, $x^u = [10 \ 10]$.

```

>> delf = @(x) [-1.2; -3]; delg = @(x) [2*x(1); 12*x(2)];
>> x0 = [1 0]; xl = [0 0]; xu = [10 10]; nc = 0; ncs = 1; crit = 1e-4; kmax =
1e3;
>> [xopt, fopt, iter] = zoutopt(@funz, delf, delg, x0, xl, xu, nc, ncs, crit, kmax)
xopt =
0.6998 0.2916
fopt =
-1.7146
iter =
5

```

10.3.3 GENERALIZED REDUCED GRADIENT (GRG) METHOD

The generalized reduced gradient (GRG) method handles nonlinear equality constraints. Consider a minimization problem with nonlinear constraints of the form

$$\begin{aligned}
 & \text{Minimize } f(x) \\
 & \text{Subject to } h_i(x) \leq 0 \quad (i = 1, \dots, m), \quad l_k(x) = 0 \quad (k = 1, \dots, l), \quad x_j^L \leq x_j \leq x_j^U \quad (j = 1, \dots, n)
 \end{aligned}$$

where $x = [x_1, x_2, \dots, x_n]^T$ is a column vector of n real-valued variables. By adding a nonnegative slack variable to each of the inequality constraints, the problem can be rewritten as

$$\begin{aligned} & \text{Minimize } f(x) \\ & \text{Subject to } g_i(x) = 0 \quad (i = 1, \dots, m+l), \quad x_j^L \leq x_j \leq x_j^U \quad (j = 1, \dots, n+m) \end{aligned}$$

The GRG method is based on the elimination of variables using the equality constraints. Usually $n+m$ design variables are divided into two sets (independent variables set $[Y]$ and dependent variables set $[Z]$) as

$$X = \begin{bmatrix} Y \\ Z \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_{n-l} \end{bmatrix}, \quad Z = \begin{bmatrix} z_1 \\ \vdots \\ z_{m+l} \end{bmatrix}$$

The first derivatives of the objective and constraint functions are given by¹³

$$\begin{aligned} df(x) &= \sum_{i=1}^{n-l} \frac{\partial f}{\partial y_i} dy_i + \sum_{i=1}^{m+l} \frac{\partial f}{\partial z_i} dz_i = \nabla_Y^T f dY + \nabla_Z^T f dZ \\ dg_i(x) &= \sum_{i=1}^{n-l} \frac{\partial g_i}{\partial y_i} dy_i + \sum_{i=1}^{m+l} \frac{\partial g_i}{\partial z_i} dz_i \end{aligned}$$

The last relation can be expressed as $dg = C dY + B dZ$. When Y is held fixed, $g(x) + dg(x) = 0$. Substitution of dg into this equation gives

$$dZ = B^{-1} \{ -g(x) - C dY \}$$

If the constraints are satisfied at the vector x , $g(x) = 0$ and $dZ = -B^{-1}C dY$. This equation can be rewritten as

$$d_z = -B^{-1}C d_y$$

The direction vector d is defined as $d = [d_z \ d_y]^T$. Substitution of d_z to the derivative of the objective function yields

$$df(x) = \nabla_Y^T f dY + \nabla_Z^T f dZ = (\nabla_Y^T f - \nabla_Z^T f B^{-1}C) d_y$$

The coefficient matrix is referred to as the generalized reduced gradient G_R :

$$G_R = \nabla_Y^T f - (B^{-1}C)^T \nabla_Z^T f$$

The new point is obtained as

$$x^{k+1} = x^k + \alpha_k d$$

The generalized reduced gradient method can be summarized as follows¹⁴:

1. Select a starting point x^0 that satisfies $g(x^0) = 0$ with $x^L \leq x^0 \leq x^U$.
2. Determine the basic and nonbasic variables and calculate the reduced gradient G_R .
3. Find the direction vector d . If $\|d\| \leq \epsilon$, stop the procedure (ϵ : very small number).

4. Calculate the current step size α_k and determine x^{k+1} .
5. Check the feasibility of the new point x^{k+1} : if this point is feasible, set $x^k = x^{k+1}$ and go to step 2. Otherwise, compute a corrected point x_c^{k+1} . If $f(x_c^{k+1}) < f(x^k)$, set $x^k = x_c^{k+1}$ and go to step 2. If $f(x_c^{k+1}) > f(x^k)$, set $\alpha_k = \alpha_k/2$, calculate x^{k+1} , and repeat step 5.

The MATLAB function *grgopt* implements the generalized reduced gradient method. The basic syntax is

```
[xopt, fopt, iter] = grgopt(grgfun, delgrgf, delgrgg, x0, xl, xu, kmax, crit)
```

where *grgfun* is the objective function and constraints, *delgrgf* is the gradient of the objective function, *delgrgg* is the gradient of constraints, *x0* is the starting point vector, *xl* and *xu* are the lower and upper bounds on *x*, *kmax* is the number of maximum possible iterations, *crit* is the stopping criterion, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

```
function [xopt, fopt, iter] = grgopt(grgfun, delgrgf, delgrgg, x0, xl, xu, kmax, crit)
% grgopt.m: minimization by the generalized reduced gradient (GRG) method
% Problem type: min. f(x)
% subject to gj(x) = 0 (j=1,...,nca), xl <= x <= xu
% Inputs:
%   grgfun: objective function and constraints
%   delgrgf: gradient of f
%   delgrgg: gradient of g
%   x0: starting point (must satisfy all constraints)
%   xl,xu: lower and upper limit of x
%   kmax: maximum possible iterations
%   crit: stopping criterion
% Outputs:
%   xopt: optimal point
%   fopt: function value at the optimal point (=f(xopt))
%   iter: number of iterations
% Example:
%   (objective function and constraints are defined by function fg1)
%   df1 = @(x) [-1.2 -3 0]; dg1 = @(x) [2*x(1) 12*x(2) 1];
%   x0 = [0 0 1]; xl = [0 0 0]; xu = [10 10 10]; crit = 1e-4; kmax = 1e3;
%   [xopt, fopt, iter] = grgopt(@fg1,df1,dg1,x0,xl,xu,kmax,crit)

dcrit = crit/10; fcrit = crit*1e-2; x = x0; [f g] = grgfun(x); [ncs nv] =
size(delgrgg(x));
for k = 1: ncs
    if (abs(g(k)) > crit), disp('Infeasible starting point.'), break; end
end
indc = 0; fold = f;
for iter = 1:kmax
    df = delgrgf(x); dg = delgrgg(x);
    for k = 1:nv, nbr(k) = k; end
    [nbr,df,dg] = redgr(nbr,nv,ncs,x,xl,xu,df,dg, crit); % basic variables
    dm = 0.;
    for k = ncs+1:nv
        nk = nbr(k); d(nk) = -df(nk);
        if (x(nk) < xl(nk)+crit & df(nk) > 0), d(nk) = 0; end
        if (x(nk) > xu(nk)-crit & df(nk) < 0), d(nk) = 0; end
        if (dm < abs(d(nk))) dm = abs(d(nk)); end
    end
end
```

```

if (dm < crit), break; end
for k = 1: ncs
    nk = nbr(k); d(nk) = 0;
    for j = ncs+1: nv, nj = nbr(j); d(nk) = d(nk) - dg(k, nj)*d(nj);
    end
end
x0 = x; alpha = findstep(nv, x, xl, xu, d); x = x0;
for k = 1: nv, xtemp(k) = x(k) + alpha*d(k); end
[ftemp g] = grgfun(xtemp); df = delgrgf(xtemp); dg = delgrgg(xtemp);
fup = 0;
for k = 1: nv, fup = fup + df(k)*d(k); end
if (fup > 0 | ftemp > f)
    c = 0; x0 = x; [alpha, ftemp] = goldsec(nv, ncs, c, alpha, dcrit, d, x);
    x = x0;
    for k = 1: nv, xtemp(k) = x(k) + alpha*d(k); end
end
[c g] = grgfun(xtemp); ag = abs(g(1));
for k = 1: ncs
    if (ag < abs(g(k))), ag = abs(g(k)); end
end
% If infeasible, apply Newton's method
if (ag > crit)
    blim = 20;
    for bi0 = 1: blim
        bnew = 0; bi2 = 12;
        for bi1 = 1: bi2
            bnew = bnew + 1;
            df = delgrgf(xtemp); dg = delgrgg(xtemp);
            [g] = rednewton (dg, g, nbr, ncs); bf = 0;
            for k = 1: ncs
                nk = nbr(k); xtemp(nk) = xtemp(nk) - g(k);
                if ((xtemp(nk) < xl(nk)) | (xtemp(nk) > xu(nk))), bf
                = 1; break; end
            end
            if (bf == 1), break; end
            [c g] = grgfun(xtemp); ag1 = abs(g(1));
            for k = 1: ncs
                if (ag1 < abs(g(k))), ag1 = abs(g(k)); end
            end
            if (bf == 0 & ag1 < crit), break; end
            bf = 1;
            if (bnew > 3 | ag1 > ag), break; end
            ag = ag1;
        end
        if (bf == 0)
            [ftemp, g] = grgfun(xtemp);
            if (ftemp < f), break; end
        end
        alpha = alpha/2;
        for k = 1: nv, xtemp(k) = x(k) + alpha*d(k); end
        [c g] = grgfun(xtemp); ag = abs(g(1));
        for k = 1: ncs
            if (ag < abs(g(k))), ag = abs(g(k)); end
        end
    end
end

```

```

if (bf==0 & ftemp<f)
    for k = 1: nv, x(k) = xtemp(k); end
    f = ftemp;
end
fcrit = abs(f)*fcrit + fcrit;
if (abs(f - fold) < fcrit)
    indc = indc + 1;
    if (indc > 1), break; end
else
    indc = 0;
end
fold = f;
end

xopt = x; fopt = f;
end

%-----
function [x4,ft] = goldsec(nv,ncs,x1,x4,dcrit,d,x)
tau = (sqrt(5)-1)/2; x2 = tau*x1 + (1-tau)*x4;
for k = 1:nv, xtemp(k) = x(k) + x2*d(k); end
[f2, g] = grgfun(xtemp);
for count = 1:100
    x3 = tau*x4 + (1-tau)*x1;
    for k = 1:nv, xtemp(k) = x(k) + x3*d(k); end
    [f3, g] = grgfun(xtemp);
    if (f2 < f3), x4 = x1; x1 = x3;
    else, x1 = x2; x2 = x3; f2 = f3;
    end
    if (abs(x4 - x1) <= dcrit), break; end
end
x4 = x2; ft = f2;
end

%-----
function [g] = rednewton(dgr, g, nbr, ncs)
for k = 1: ncs-1
    nk = nbr(k);
    for jk = k+1:ncs
        c = dgr(jk,nk)/dgr(k,nk);
        for j = k + 1: ncs, nj = nbr(j); dgr(jk,nj) = dgr(jk,nj) -
            c*dgr(k,nj); end
        g(jk) = g(jk) - c*g(k);
    end
end
g(ncs) = g(ncs)/dgr(ncs,nbr(ncs));
for jm = 1:ncs-1
    jk = ncs - jm; im = nbr(jk); c = 1/dgr(jk,im); g(jk) = c*g(jk);
    for k = jk + 1: ncs, nk = nbr(k); g(jk) = g(jk) - c*dgr(jk,nk)*g(k); end
end
end

%-----
function [nbr,df,dg] = redgr(nbr,nv,ncs,x,xl,xu,df,dg,xcrit)
for k = 1: ncs
    nk = nbr(k); kcount = 0;
    for j = k: nv
        nj = nbr(j);
        if (x(nj) > xl(nj)+xcrit & x(nj) < xu(nj)-xcrit)

```

```

kcount = kcount + 1;
if (kcount == 1), pivot = dg(k,nj); jpv = j;
else
    if (abs(pivot) < abs(dg(k,nj))), pivot = dg(k,nj); jpv =
        j; end
    end
end
nbr(k) = nbr(jpv); nbr(jpv) = nk; nk = nbr(k);
if (k == ncs), break; end
for jk = k+1: ncs
    dgratio = dg(jk,nk)/dg(k,nk);
    for j = k+1:nv, nj = nbr(j); dg(jk,nj) = dg(jk,nj) -
        dgratio*dg(k,nj); end
end
nbs = nbr(ncs);
for j = ncs + 1: nv
    nj = nbr(j); dg(ncs,nj) = dg(ncs,nj)/dg(ncs,nbs);
    for jm = 1:ncs-1
        jk = ncs-jm; km = nbr(jk); dgratio = 1/dg(jk,km);
        dg(jk,nj) = dgratio*dg(jk, nj);
        for k = jk+1: ncs, nk = nbr(k); dg(jk,nj) = dg(jk,nj) -
            dgratio*dg(jk,nk)*dg(k,nj); end
    end
end
% Calculate reduced gradient
for jk = ncs+1:nv
    km = nbr(jk);
    for j = 1: ncs, nj = nbr(j); df(km) = df(km) - dg(j,km)*df(nj); end
end
end

-----
function [alpha] = findstep(nv, x, xl, xu, d)
% Calculate maximum step size
kcount = 0;
for j = 1:nv
    au = xu(j) - x(j); % check upper bound
    if (d(j) > 1E-30*(au+1))
        kcount = kcount + 1; ad = au/d(j);
        if (kcount == 1), alpha = ad; end
        if (kcount > 1)
            if (alpha > ad), alpha = ad; end
        end
    end
    al = x(j) - xl(j); % check lower bound
    if (-d(j) > 1E-30*(al + 1))
        kcount = kcount + 1;
        ad = -al/d(j);
        if (kcount == 1), alpha = ad; end
        if (kcount > 1)
            if (alpha > ad), alpha = ad; end
        end
    end
end
end
end

```

Example 10.14: The Generalized Reduced Gradient Method

Find the solution of the following constrained minimization problem using the generalized reduced gradient method:

$$\text{Minimize } f(x) = x_1^2 + x_2^2 + 2.3x_3^2 - 1.2x_4^2 - 4x_1 - 6x_2 - 20x_3 + 6x_4 + 100$$

Subject to

$$h_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_1 - x_2 + x_3 - x_4 \leq 7$$

$$h_2(x) = x_1^2 + 2x_2^2 + x_3^2 + 2x_4^2 - x_1 - x_4 \leq 11$$

$$h_3(x) = 2x_1^2 + x_2^2 + x_3^2 + 2x_1 - x_2 - x_4 \leq 6, \quad -100 \leq x_1, x_2, x_3, x_4 \leq 100$$

The inequality constraints can be converted to equalities by introducing slack variables as follows:

$$g_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_1 - x_2 + x_3 - x_4 + x_5 - 7 = 0$$

$$g_2(x) = x_1^2 + 2x_2^2 + x_3^2 + 2x_4^2 - x_1 - x_4 + x_6 - 11 = 0$$

$$g_3(x) = 2x_1^2 + x_2^2 + x_3^2 + 2x_1 - x_2 - x_4 + x_7 - 6 = 0$$

As a starting point, we can use $x^0 = [0 \ 0 \ 0 \ 0 \ 7 \ 11 \ 6]$. Note that this point satisfies all equality constraints. Set $kmax = 1000$ and $crit = 1 \times 10^{-4}$.

Solution

The objective function and the equality constraints can be defined as a function as follows:

```
function [f g] = grgfun(x)
x1 = x(1); x2 = x(2); x3 = x(3); x4 = x(4);
x5 = x(5); x6 = x(6); x7 = x(7); % slack variables
f = x1^2 + x2^2 + 2.3*x3^2 - 1.2*x4^2 - 4*x1 - 6*x2 - 20*x3 + 6*x4 + 100;
g(1) = x1^2 + x2^2 + x3^2 + x4^2 + x1 - x2 + x3 - x4 + x5 - 7;
g(2) = x1^2 + 2*x2^2 + x3^2 + 2*x4^2 - x1 - x4 + x6 - 11;
g(3) = 2*x1^2 + x2^2 + x3^2 + 2*x1 - x2 - x4 + x7 - 6;
end
```

The gradients of $f(x)$ and $g(x)$ are given by

$$\frac{\partial f}{\partial x_1} = 2.3x_1 - 4, \quad \frac{\partial f}{\partial x_2} = 2x_2 - 6, \quad \frac{\partial f}{\partial x_3} = 4.6x_3 - 20, \quad \frac{\partial f}{\partial x_4} = -2.4x_4 + 6$$

$$\nabla g = \begin{bmatrix} \nabla g_1 \\ \nabla g_2 \\ \nabla g_3 \end{bmatrix} = \begin{bmatrix} 2x_1 + 1 & 2x_2 - 1 & 3x_3 + 1 & 2x_4 - 1 & 1 & 0 & 0 \\ 2x_1 - 1 & 4x_2 & 2x_3 & 4x_4 - 1 & 0 & 1 & 0 \\ 4x_1 + 2 & 2x_2 - 1 & 2x_3 & -1 & 0 & 0 & 1 \end{bmatrix}$$

The functions $delgrgf$ and $delgrgg$ define the gradients of $f(x)$ and $g(x)$, respectively:

```
function df = delgrgf(x)
x1 = x(1); x2 = x(2); x3 = x(3); x4 = x(4);
df = zeros(1,length(x));
df(1) = 2.3*x1-4; df(2) = 2*x2-6; df(3) = 4.6*x3-20; df(4) = -2.4*x4+6;
end
```

```

function dg = delgrgg(x)
x1 = x(1); x2 = x(2); x3 = x(3); x4 = x(4);
dg = [2*x1+1 2*x2-1 3*x3+1 2*x4-1 1 0 0;
      2*x1-1 4*x2 2*x3 4*x4-1 0 1 0;
      4*x1+2 2*x2-1 2*x3 -1 0 0 1];
end

```

The following commands execute the function *grgopt* to find the solution:

```

>> x0 = [0 0 0 0 7 11 6]; xl = [-100 -100 -100 -100 0 0 0]; xu =
100*ones(1,7); kmax = 1e3; crit = 1e-4;
>> [xopt,fopt,iter] = grgopt(@grgfun,@delgrgf,@delgrgg,x0,xl,xu,kmax,crit)
xopt =
0.0081 0.4922 0.6609 0.2043 6.3066 10.2076 6.0011
fopt =
86.2196
iter =
5

```

10.3.4 SEQUENTIAL QUADRATIC PROGRAMMING (SQP) METHOD

Consider a constrained minimization

Minimize $f(x)$

Subject to $v_i(x) = 0$ ($i = 1, \dots, p$), $w_k(x) \geq 0$ ($k = 1, \dots, q$)

where $x = [x_1, x_2, \dots, x_n]^T$ is a column vector of n real-valued variables. Let $\{x^k, \lambda^k, \mu^k\}$ be the k th iteration and $\{\delta_x, \delta_\lambda, \delta_\mu\}$ be a set of increment vectors such that the following Karush–Kuhn–Tucker (KKT) conditions are satisfied at the next iteration $\{x^{k+1}, \lambda^{k+1}, \mu^{k+1}\} = \{x^k + \delta_x, \lambda^k + \delta_\lambda, \mu^k + \delta_\mu\}$:

$$\nabla_x \mathcal{L}(x, \lambda, \mu) = 0, \quad v_i(x) = 0 \quad (i = 1, \dots, p), \quad w_k(x) \geq 0 \quad (k = 1, \dots, q)$$

$$\mu \geq 0, \quad \mu^k w_k(x) = 0 \quad (k = 1, \dots, q)$$

where $\mathcal{L}(x, \lambda, \mu)$ is the Lagrangian defined as

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \sum_{i=1}^p \lambda_i v_i(x) - \sum_{k=1}^q \mu^k w_k(x)$$

We obtain the approximate KKT conditions as¹⁵

$$Z^k \delta_x + g^k - A_{ek}^T \lambda^{k+1} - A_{ik}^T \mu^{k+1} = 0, \quad A_{ek} \delta_x = -v_k, \quad A_{ik} \delta_x \geq -w_k, \quad \mu^{k+1} \geq 0$$

$$(\mu^{k+1})_j (A_{ik} \delta_x + w_k)_j = 0 \quad (j = 1, 2, \dots, q)$$

where

$$Z^k = \nabla_x^2 f(x^k) - \sum_{i=1}^p (\lambda^k)_i \nabla_x^2 v_i(x^k) - \sum_{j=1}^q (\mu^k)_j \nabla_x^2 w_i(x^k), \quad g^k = \nabla_x f(x^k),$$

$$A_{ek} = \begin{bmatrix} \nabla_x^T v_1(x^k) \\ \vdots \\ \nabla_x^T v_p(x^k) \end{bmatrix}, \quad A_{ik} = \begin{bmatrix} \nabla_x^T w_1(x^k) \\ \vdots \\ \nabla_x^T w_q(x^k) \end{bmatrix}$$

Given $\{x^k, \lambda^k, \mu^k\}$, the approximate KKT conditions may be interpreted as the exact KKT conditions of the quadratic programming problem given by

$$\text{Minimize } h(x) = \frac{1}{2} \delta^T Z^k \delta + \delta^T g^k$$

$$\text{Subject to } A_{ek} \delta = -v_k, A_{ik} \delta \geq -w_k$$

If δ_x is a regular solution of this quadratic programming problem, the approximate KKT condition can be written as

$$Z^k \delta_x + g^k - A_{ek}^T \lambda^{k+1} - A_{ak}^T \hat{\mu}^{k+1} = 0$$

where the matrix A_{ak} consists of those rows of A_{ik} that satisfy the equality $(A_{ik} \delta_x + w_k)_j = 0$ and $\hat{\mu}^{k+1}$ is the associated Lagrange multiplier. If δ_x is the solution of the above quadratic programming problem, the current point x^k can be updated by

$$x^{k+1} = x^k + \alpha_k \delta_x$$

The value of α_k is calculated as¹⁶

$$\alpha_k = 0.95 \min \{ \alpha_1^*, \alpha_2^* \}, \quad \alpha_1^* = \arg \left[\min_{0 \leq \alpha \leq 1} \varphi(\alpha) \right], \quad \alpha_2^* = \max \left\{ \alpha : w_j(x^k + \alpha \delta_x) \geq 0, j \in J \right\}$$

$$\varphi(\alpha) = f(x^k + \alpha \delta_x) + \beta \sum_{k=1}^p v_i^2(x^k + \alpha \delta_x) - \sum_{j=1}^q (\mu^{k+1})_j w_j(x^k + \alpha \delta_x)$$

where $J = \{j : (\mu^{k+1})_j = 0\}$. The Hessian Z^k can be updated by using the BFGS formula as

$$Z^{k+1} = Z^k + \frac{\eta^k (\eta^k)^T}{\delta_x^T \eta^k} - \frac{Z^k \delta_x \delta_x^T Z^k}{\delta_x^T Z^k \delta_x}, \quad \eta^k = \theta \gamma^k + (1 - \theta) Z^k \delta_x$$

$$\gamma^k = (g^{k+1} - g^k) - (A_{e,k+1} - A_{e,k})^T \lambda^{k+1} - (A_{i,k+1} - A_{i,k})^T \mu^{k+1}$$

$$\theta = \begin{cases} 1: & \delta_x^T \gamma^k \geq 0.2 \delta_x^T Z^k \delta_x \\ \frac{0.8 \delta_x^T Z^k \delta_x}{\delta_x^T Z^k \delta_x - \delta_x^T \gamma^k}: & \text{otherwise} \end{cases}$$

The sequential quadratic programming algorithm can be summarized as follows:

1. Select x^0 and μ^0 such that $w_k(x^0) \geq 0$ ($k = 1, \dots, q$) and $\mu^0 \geq 0$. Set $k = 0$ and $Z^0 = I_n$.
2. Calculate g^k, A_{ek}, A_{ik}, v^k , and w^k .

3. Solve the quadratic programming problem for δ_x , and calculate the Lagrange multiplier λ^{k+1} , $\hat{\mu}^{k+1}$, and α_k .
4. Set $\delta_x = \alpha_k \delta_x$ and $x^{k+1} = x^k + \delta_x$.
5. If $\|\delta_x\| \leq \epsilon$, stop the calculation procedure.
6. Calculate γ^k , θ , η^k , and Z^{k+1} . Set $k = k + 1$ and go to step 2.

The MATLAB function *sqpopt* implements the sequential quadratic programming method. The basic syntax is

```
[xopt, fopt, iter] = sqpopt(fun, dfun, x0, lam0, mu0, crit)
```

where *fun* is the objective function and constraints, *dfun* is the gradient of these functions, *x0* is the starting point vector, *lam0* and *mu0* are the initial values of Lagrangian multipliers, *crit* is the stopping criterion, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

```
function [xopt, fopt, iter] = sqpopt(fun, dfun, x0, lam0, mu0, crit)
% sqpopt.m: minimization by using SQP algorithm
% Problem type: minimize f(x) subject to h(x) = 0, g(x) >= b
% Inputs:
%   fun: objective and constraint functions
%   dfun: gradients of the objective and constraint functions
%   x0: starting point
%   lam0, mu0: initial Lagrange multipliers
%   crit: stopping criterion
% Outputs:
%   xopt: optimal point
%   fopt: objective function value at x=xopt
%   iter: number of iterations
% Example:
% x0 = [3 3]'; lam0 = 1; mu0 = 4; crit = 1e-6;
% [xopt, fopt, iter] = sqpopt(@fun, @dfun, x0, lam0, mu0, crit)

x = x0(:); n = length(x); % number of variables
lam1 = lam0 + 1; Hj = eye(n); fv = fun(x); aj = fv(2:lam1); cj =
fv((lam1+1):(lam1+mu0));
Gj = dfun(x); gj = Gj(:,1); Aej = Gj(:,2:lam1)'; Aij =
Gj(:,(lam1+1):(lam1+mu0))'; iter = 0; d = 1;
while d >= crit
    delx = quadpr(Hj, gj, Aej, -aj, Aij, -cj, zeros(n,1), crit);
    ad = Aij * (x+delx) + cj; k = find(ad <= crit); nk = length(k); muj =
    zeros(mu0,1);
    if nk == 0, lamj = inv(Aej * Aej') * Aej * (Hj * delx + gj);
    else
        Aaik = Aij(k,:); Aaj = [Aej; Aaik]; mun = inv(Aaj * Aaj') * Aaj * (Hj *
        delx + gj);
        lamj = mun(1:lam0); mujh = mun(lam1:end); muj(k) = mujh;
    end
    alpha = linsearch(fun, x, delx, lam1, muj, crit); delx = alpha * delx; x = x +
    delx; grd = dfun(x);
    grd1 = grd(:,1); Agrd = grd(:,2:lam1)'; Am = grd(:,(lam1+1):(lam1+mu0))';
```

```

gamj = (grd1-gj)-(Agrd-Aej)'*lamj-(Am-Aij)'*muj; qj = Hj*delx; dg =
delx'*gamj; dq = delx'*qj;
if dg >= 0.2*dq, theta = 1;
else, theta = 0.8*dq/(dq-dg);
end
eta = theta*gamj + (1-theta)*qj; invdq = 1/dq; invde = 1/(delx'*eta);
Hj = Hj + invde*(eta*eta') - invdq*(qj*qj');
Aej = Agrd; Aij = Am; gj =
grd1; fv = fun(x);
aj = fv(2:lam1); cj = fv((lam1+1):(lam1+mu0)); d = norm(delx); iter =
iter + 1;
end
xopt = x; fopt = fv(1);
end

function alpha = linsearch(fun,xj,dx,lam,muj,crit)
% Determine step size by line search method
nmuj = length(muj); alrange = 0:0.01:1; nint = length(alrange);
hz = zeros(nint,1);
for j = 1:nint
    xdj = xj + alrange(j)*dx; fv = fun(xdj); af = fv(2:lam); cf =
    fv((lam+1):(lam+nmuj));
    hz(j) = fv(1) + 1e-2*sum(af.^2) - muj'*cf;
end
[mval,indhz] = min(hz); atemp = alrange(indhz); indmu = find(muj <=
crit); mc = length(indmu);
if mc == 0, alpha = 0.95*atemp;
else
    dv = zeros(mc,1);
    for k = 1:mc
        for j = 1:nint
            aj = alrange(j); xdj = xj + aj*dx; fcj = fun(xdj);
            cj = fcj((lam+1):(lam+nmuj)); hz(j) = cj(indmu(k));
        end
        indhz = find(hz < 0); hc = length(indhz);
        if hc == 0, dv(k) = 1;
        else, dv(k) = alrange(indhz(1)-1);
        end
    end
    mdv = min(dv); alpha = 0.95*min(atemp,mdv);
end
end

function xqr = quadpr(Q,c,Aeq,beq,Ane,bne,x0,crit)
% minimization by quadratic programming method
nbeg = length(beq); nbne = length(bne);
rnbne = nbne + 1.5*sqrt(nbne); aone = 1-crit; x = x0(:); y = Ane*x - bne;
zeta = zeros(nbeg,1); gama = ones(nbne,1); ym = y.*gama; sumy = sum(ym);
iter = 0;
while sumy > crit
    tau = sumy/rnbne; resid = -Q*x - c + Aeq'*zeta + Ane'*gama;
    diffr = beq - Aeq*x; numt = tau - y.*gama; yj = gama./y; ysj =
    numt./y; ydm = diag(yj);
    Gr = inv(Q + Ane'*ydm*Ane); ag = Aeq*Gr*Aeq';
    ayj = resid + Ane'*ysj; diffag = diffr - Aeq*Gr*ayj; delz =
    inv(ag)*diffag;
    dx = Gr*(ayj + Aeq'*delz); dy = Ane*dx; dgama = (numt - (gama.*dy))./y;

```

```

indny = find(dy < 0); yr = min(y(indny)./(-dy(indny)));
indny = find(dgama < 0); gamr = min(gama(indny)./(-dgama(indny)));
aj = aone*min([1 yr gamr]); x = x + aj*dx; gama = gama + aj*dgama;
zeta = zeta + aj*delz;
y = Ane*x - bne; ym = y.*gama; sumy = sum(ym);
iter = iter + 1;
end
xqr = x;
end

```

Example 10.15: The Sequential Quadratic Programming Method

Find the solution of the following constrained minimization problem using the sequential quadratic programming method:

$$\begin{aligned}
 \text{Minimize } f(x) &= 2x_1 + x_2^2 \\
 \text{Subject to } h(x) &= x_1^2 + x_2^2 = 8, \quad 0 \leq x_1 \leq 4, \quad 1 \leq x_2 \leq 5
 \end{aligned}$$

As a starting point, use $x^0 = [2, 3]^T$. The initial values of Lagrangian multipliers can be set as 1 and 3. Set crit = 1×10^{-6} .

Solution

The inequality constraints can be rearranged as follows:

$$x_1 \geq 0, \quad -x_1 + 4 \geq 0, \quad x_2 - 1 \geq 0, \quad -x_2 + 5 \geq 0$$

The function *fun* defines the objective function, equality constraint, and inequality constraints:

```

function fv = fun(x)
fv = [2*x(1) + x(2)^2]; % objective function
x(1)^2 + x(2)^2 - 8; % equality constraint: h(x) = 0
x(1); % inequality constraint: g1(x) >= 0
-x(1) + 4; % inequality constraint: g2(x) >= 0
x(2) - 1; % inequality constraint: g3(x) >= 0
-x(2) + 5]; % inequality constraint: g4(x) >= 0
end

```

Gradients of the objective function, equality constraint, and inequality constraints are given by

$$\nabla f(x) = [2 \quad 2x_2], \quad \nabla h(x) = [2x_1 \quad 2x_2], \quad \nabla g(x) = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}$$

The function *dfun* defines these gradients as follows:

```

function dfv = dfun(x)
df = [2 2*x(2)]; % gradient of objective function
dh = [2*x(1) 2*x(2)]; % gradient of equality constraint (h(x) = 0)
dg = [1 0;-1 0;0 1;0 -1]; % gradient of inequality constraint (gi(x) >= 0)
dfv = [df' dh' dg'];
end

```

Note that each gradient vector is converted to a column vector and that the gradient matrix for ∇g is transposed. The following commands implement the SQP algorithm to get solutions:

```
>> x0 = [2 3]'; lam0 = 1; mu0 = 3; crit = 1e-6;
>> [xopt,fopt,iter] = sqpopt(@fun,@dfun,x0,1am0,mu0,crit)
xopt =
    2.6458
    1.0000
fopt =
    6.2915
iter =
    8
```

10.4 DIRECT SEARCH METHODS

10.4.1 CYCLIC COORDINATE METHOD

In the cyclic coordinate search method, the search is performed along each of the coordinate directions for finding the minimum. Let u_k be the unit vector along the coordinate direction k . Then the value of α_k that minimizes $f(\alpha) = f(x + \alpha u_k)$ is determined, and a move is made to the new point $x + \alpha_k u_k$ at the end of the search along the direction k . The search is conducted along all the possible directions as one stage. The search for the minimum terminates at the point where the gradient appears to be zero. For n -dimensional problem, the cyclic coordinate algorithm can be summarized as follows¹⁷:

1. Choose a starting point x^1 and calculate $f_1 = f(x^1)$. Set $k = 1$, $x = x^1$, and $f = f_1$.
2. Determine α_k that minimizes $f(\alpha) = f(x + \alpha u_k)$.
3. Modify the current point by setting $x = x + \alpha_k u_k$ and calculate $f = f(x)$.
4. If $k < n$, set $k = k + 1$ and go to step 2.
5. Specify the direction as $d = x - x^1$ and determine α_d that minimizes $f(x + \alpha d)$.
6. Move to the new point by setting $x = x + \alpha_d d$ and calculate $f = f(x)$.
7. If $\|d\| > \epsilon$ or $|f - f_1| > \epsilon$, set $x^1 = x$, $f_1 = f$ and go to step 2.

The MATLAB function *cycopt* implements the cyclic coordinate search method. The basic syntax is

```
[xopt,fopt,iter] = cycopt(fcyc,x0,crit)
```

where *fcyc* is the objective function, *x0* is the starting point vector, *crit* is the stopping criterion, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

```
function [xopt,fopt,iter] = cycopt(fcyc,x0,crit)
% cycopt.m: minimization by the cyclic coordinate search
% Inputs:
%   fcyc: objective functions
%   x0: starting point
%   crit: stopping criterion
% Outputs:
%   xopt: optimal point
%   fopt: objective function value at x=xopt
%   iter: number of iterations
% Example:
% x0 = [-3 -1 0 1]; crit = 1e-4;
% fc = @(x) (x(1)+10*x(2))^2+5*(x(3)-x(4))^2+(x(2)-2*x(3))^4+10*(x(1)-x(4))^4;
% [xopt,fopt,iter] = cycopt(fc,x0,crit)
```

```

stsiz = 0.1; x = x0; xk = x; nv = length(x); fk = fcyc(xk); fold = fk;
iter = 0;
while (1)
    iter = iter + 1;
    for j = 1:nv, h(j) = 0; end
    for k = 1:nv
        for j = 1:nv, d(j) = 0; end
        d(k) = 1; aut = 0;
        [stfit,fvfit] = quadfit(fcyc,aut,fk,stsiz,xk,d,crit);
        xk(k) = xk(k) + stfit; h(k) = stfit; fk = fvfit;
    end
    aut = 0; [stfit,fvfit] = quadfit(fcyc,aut,fk,stsiz,xk,h,crit);
    for j = 1:nv, xk(j) = xk(j) + stfit * h(j); end
    if (abs(fk-fold) <= crit), break; end
    fold = fk;
end
xopt = xk; fopt = fcyc(xk);
end

function [x2, f2] = quadfit(fcyc,x1,f1,stsiz,x0,d,crit)
fr = 0.05; % 0 < sfrac < 0.5
[x1,x2,x3,f1,f2,f3] = approx3pt(fcyc,x1,f1,stsiz,x0,d); % 3-point pattern
tau = (sqrt(5) - 1)/2; redi = 2;
if (x3 < x1)
    xtemp = x1; ftemp = f1; x1 = x3; f1 = f3; x3 = xtemp; f3 = ftemp;
end
iflag = 0; indc = 0; jndc = 0;
while (1)
    xdold = abs(x3 - x1); favg = (f1 + f2 + f3)/3.;
    if iflag == 0
        A = (x1-x2)*(x1-x3); B = (x2-x1)*(x2-x3); C = (x3-x1)*(x3-x2);
        x4 = (f1*(x2+x3)/A + f2*(x1+x3)/B + f3*(x1+x2)/C)/(f1/A+f2/
        B+f3/C)/2;
    else
        if x2 <= (x1+x3)/2, x4 = x2 + (1-tau)*(x3-x2);
        else, x4 = x3 - (1-tau)*(x2-x1);
        end
        iflag = 0;
    end
    delt = fr*min(abs(x2-x1),abs(x3-x2));
    if abs(x4-x1) < delt, x4 = x1+delt;
    elseif abs(x4 - x3) < delt, x4 = x3 - delt;
    elseif abs(x4-x2) < delt
        if x2 > (x1 + x3)/2, x4 = x2-delt;
        else, x4 = x2+delt;
        end
    end
    f4 = fcyc(x0 + x4*d);
    if (x4 > x2)
        if (f4 >= f2), x3 = x4; f3 = f4;
        else, x1 = x2; f1 = f2; x2 = x4; f2 = f4;
        end
    else
        if (f4 >= f2), x1 = x4; f1 = f4;
        else, x3 = x2; f3 = f2; x2 = x4; f2 = f4;
        end
    end
end

```

```

xdnew = abs(x3-x1); fvnew = (f1 + f2 + f3)/3. ;
if abs(x3 - x1) <= crit, break; end
if abs(fvnew - favg) <= crit
    jndc = jndc + 1;
    if jndc == 2, break; end
else, jndc = 0;
end
if xdnew/xdold > tau
    indc = indc + 1;
    if indc == redi, indc = 0; iflag = 1; end
else, indc = 0; iflag = 0;
end
end
end

function [x1,x2,x3,f1,f2,f3] = approx3pt(fcyc,x1,f1,stsize,x0,d)
% Three-point pattern
tau = (sqrt(5) - 1)/2; stlen = stsize; x2 = x1 + stlen; x = x0 + x2*d; f2 =
fcyc(x);
if (f2 > f1)
    temp = x1; x1 = x2; x2 = temp; temp = f1; f1 = f2; f2 = temp; stlen =
-stlen;
end
while (1)
    stlen = stlen/tau; x3 = x2 + stlen; x = x0 + x3*d; f3 = fcyc(x);
    if (f3 > f2), break;
    else, f1 = f2; x1 = x2; f2 = f3; x2 = x3;
    end
end
end

```

Example 10.16: The Cyclic Coordinate Search Method

Find the minimum of Powell's function using the cyclic coordinate search method:

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

As a starting point, use $x^0 = [-3, -1, 0, 1]$ and $\text{crit} = 1 \times 10^{-4}$.

Solution

```

>> x0 = [-3 -1 0 1]; crit = 1e-4;
>> fc = @(x)
(x(1)+10*x(2))^2+5*(x(3)-x(4))^2+(x(2)-2*x(3))^4+10*(x(1)-x(4))^4;
>> [xopt,fopt,iter] = cycopt(fc,x0,crit)
xopt =
    0.1042    -0.0097    0.0543    0.0566
fopt =
    3.1864e-04
iter =
    14

```

10.4.2 HOOKE AND JEEVES PATTERN SEARCH METHOD

In the Hooke and Jeeves pattern search method, an initial step size α is chosen and the search is initiated from a given starting point. Let u_k be the unit vector along the coordinate direction x_k . Then the function value at $x + \alpha u_k$ is evaluated where α is the step size. If the function reduces, the current

point x is updated according to $x + \alpha u_k$. If the function value does not decrease, calculate $f(x - \alpha u_k)$. If this function value reduces, x is updated to be $x - \alpha u_k$.

Suppose that the starting point is denoted by Q whose coordinate is x_Q . If the search is not successful about the starting point Q , the step size is reduced as $\alpha = c\alpha$ where $c(1 < c < 1)$ is a step reduction parameter. If the exploration is successful, the pattern direction $x_R - x_Q$ is established and the point R becomes the new starting point. The Hooke and Jeeves pattern search algorithm can be summarized as follows¹⁸:

1. Select a starting point x^1 , an initial step size α , a step reduction parameter c , and an acceleration parameter β .
2. Set $x_Q = x^1$ and search about the point x_Q . If the function value reduces, define the new point generated as a new starting point x_Q and rename the old starting point as x_Q' . If the search is not successful, reduce the step size as $\alpha = c\alpha$.
3. Find a point $x_P = x_Q + \beta(x_Q - x_Q')$ and perform exploration about the point x_P .
4. If the function value reduces, define the new point generated as a new starting point x_Q , rename the old starting point as x_Q' , and go to step 3.
5. If the search is not successful, perform exploration about the point x_Q . Reduce the step size as $\alpha = c\alpha$ until convergence is achieved. Take the new point as a new starting point and go to step 3.

The MATLAB function *hjopt* implements the Hooke and Jeeves pattern search method. The basic syntax is

```
[xopt, fopt, iter] = hjopt(fhj, x0, crit)
```

where *fhj* is the objective function, *x0* is the starting point vector, *crit* is the stopping criterion, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

```
function [xopt, fopt, iter] = hjopt(fhj, x0, crit)
% hjopt.m: minimization by the Hooke and Jeeves pattern search method
% Inputs:
%   fhj: objective functions
%   x0: starting point
%   crit: stopping criterion
% Outputs:
%   xopt: optimal point
%   fopt: objective function value at x=xopt
%   iter: number of iterations
% Example:
% x0 = [-3 -1 0 1]; crit = 1e-4;
% f = @(x) (x(1)+10*x(2))^2+5*(x(3)-x(4))^2+(x(2)-2*x(3))^4+10*(x(1)-x(4))^4;
% [xopt, fopt, iter] = hjopt(f, x0, crit)

inist = 0.5; fr = 0.125; x = x0; xb = x; n = length(x0); stsize = inist;
nc = 0; % number of contraction steps
nb = 0; % number of base changes
np = 0; % number of pattern moves
fb = fhj(xb); iter = 1;
while (1)
    fk = fb; [fk, x] = search(fhj, n, stsize, fk, x);
    if (fb-fk > crit)
        while (2)
            icv = 0;
            for j = 1:n, cp = x(j); x(j) = 2*cp - xb(j); xb(j) = cp; end
            if (fb-fk <= crit)
                break;
            else
                fb = fhj(xb);
            end
        end
    else
        break;
    end
    nc = nc + 1;
    if (nc > nb)
        break;
    end
end
xopt = x;
fopt = fk;
iter = nc;
```

```

fb = fk; fk = fhj(x); [fk, x] = search(fhj, n, stsize, fk, x);
if (fb-fk <= crit)
    x = xb;
    if (nb > 1)
        if abs(fk - fold) < crit, icv = 1; break; end
    end
    nb = nb + 1; break;
end
np = np + 1;
end
if (icv == 1), break; end
else
    fold = fk;
    if (stsize < crit), break; end
    stsize = fr*stsize; nc = nc + 1;
end
iter = iter + 1;
end
xopt = x; fopt = fk;
end

function [fk, xk] = search(fhj, n, stsize, fk, x)
for k = 1:n
    cpt = x(k); x(k) = cpt + stsize;
    f = fhj(x);
    if (f < fk), fk = f;
    else
        x(k) = cpt - stsize; f = fhj(x);
        if (f < fk)
            fk = f;
        else, x(k) = cpt;
        end
    end
end
xk = x;
end

```

Example 10.17: The Hooke and Jeeves Pattern Search Method

Find the minimum of Rosenbrock's function $f(x)$ using the Hooke and Jeeves pattern search method:

$$f(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$$

Use $x^0 = [-1, 1]$ and $\text{crit} = 1 \times 10^{-6}$.

Solution

```

>> f = @(x) 100*(x(1)^2 - x(2))^2 + (1 - x(1))^2;
>> x0 = [-1 1]; crit = 1e-6;
>> [xopt, fopt, iter] = hjopt(f, x0, crit)
xopt =
    1.0078    1.0156
fopt =
    6.0860e-05
iter =
    10

```

10.4.3 ROSENROCK'S METHOD

In Rosenbrock's method, the search is carried out in n orthogonal directions at each stage. At the next stage, new orthogonal directions are determined. Exploration of orthogonal directions presents some useful schemes for step-size decisions. Rosenbrock's method can be summarized as follows¹⁹:

1. Set the direction vectors as $\{v_1, v_2, \dots, v_n\} = \{u_1, u_2, \dots, u_n\}$ where u_i is the unit vector. Choose a starting point x^0 , initial step lengths α_i ($i = 1, \dots, n$), a step expansion parameter $\beta (>1)$, and a step reduction parameter $c (<1)$. Set $x = y = x^0$ and $k = 1$ (k : stage counter) and $j = 0$ (j : cycle counter).
2. If $j > n$, set $j = 1$. Else set $j = j + 1$. Let $f_x = f(x)$ and $y = x + \alpha_j v_j$. Evaluate the function at y : $f_y = f(y)$. If $f_y < f_x$, set $\alpha_j = \beta \alpha_j$, replace x by y ($x = y, f_x = f_y$), and go to step 3. If $f_y \geq f_x$, set $\alpha_j = -c \alpha_j$ and repeat this step.
3. Construct n linearly independent directions w_1, w_2, \dots, w_n as $w_i = \sum_{k=1}^n \alpha_k v_k$ ($i = 1, \dots, n$). Calculate the magnitude of w_1 : $\alpha = \sqrt{w_1^T w_1}$. If α is less than the stopping criterion, stop the procedure. If convergence is not achieved, new orthogonal directions are formed using the Gram–Schmidt procedure: $w_i' = w_i - \sum_{k=1}^{i-1} (w_i^T v_k) v_k, v_i = \frac{w_i'}{\text{norm}(w_i')}$.
4. Set $k = k + 1$ and carry out the next stage calculations. The new stage begins with n coordinate directions as starting vectors.

The MATLAB function *rbopt* implements Rosenbrock's method. The basic syntax is

```
[xopt, fopt, istag] = rbopt(fros, x0, crit)
```

where *fros* is the objective function, *x0* is the starting point vector, *crit* is the stopping criterion, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *istag* is the number of calculation stages.

```
function [xopt, fopt, istag] = rbopt(fros, x0, crit)
% rbopt.m: minimization by the Rosenbrock's method
% Inputs:
%   fros: objective functions
%   x0: starting point
%   crit: stopping criterion
% Outputs:
%   xopt: optimal point
%   fopt: objective function value at x=xopt
%   istag: number of stages
% Example:
% x0 = [-3 -1 0 1]; crit = 1e-4;
% f = @(x) (x(1)+10*x(2))^2+5*(x(3)-x(4))^2+(x(2)-2*x(3))^4+10*(x(1)-x(4))^4;
% [xopt, fopt, istag] = rbopt(f, x0, crit)

n = length(x0); % number of variables
mi = 7; stsize = 1; alpha = 3; beta = 0.5; xt = x0;
% Initial vectors along coordinates
for j = 1:n
    for k = 1:n, v(j,k) = 0; end
    v(j,j) = 1;
end
ft = fros(xt); istag = 0;
```

```

while (1)
    istag = istag + 1;
    % Initialization
    for j = 1:n, stp(j) = stsize; stj(j) = 0; sfv(j,1) = 0; sfv(j,2) = 0; end
    iter = 0; icont = 0; idn = 0;
    while (2)
        iter = iter + 1;
        if iter > n, iter = 1; end
        for j = 1:n, x(j) = xt(j) + stp(iter)*v(j,iter); end
        f = fros(x);
        if f < ft
            stj(iter) = stj(iter) + stp(iter); % successful
            for j = 1:n, xt(j) = x(j); end
            ft = f; idn = 0; stp(iter) = alpha*stp(iter);
            if sfv(iter,1) == 0, sfv(iter,1) = 1; icont = icont + 1; end
        else % failure
            stp(iter) = -beta*stp(iter);
            if sfv(iter,2) == 0, sfv(iter,2) = 1; icont = icont + 1; end
            idn = idn + 1;
        end
        if icont == 2*n, break; end
        if idn > 50, mi = mi + 1; return; end
    end
    if istag == 1, fold = ft;
    else
        if abs(ft - fold) < crit, mi = mi + 1; display('Convergence
            achieved.); break; end
    end
    fold = ft;
    % Construct new vectors
    for k = 1:n
        for j = 1:n, u(k,j) = 0; end
    end
    for i = 1:n
        for j = 1:n
            for k = 1:n, u(k,i) = u(k,i) + stj(j)*v(k,j); end
        end
    end
    % Orthogonalization by Gram-Schmidt procedure
    for i = 1:n
        if i > 1
            for j = 1:n, v(j,i) = 0; end
            for k = 1:i-1
                c = 0;
                for j = 1:n, c = c + u(j,i)*v(j,k); end
                for j = 1:n, v(j,i) = v(j,i) + c*v(j,k); end
            end
            for j = 1:n, u(j,i) = u(j,i) - v(j,i); end
        end
        c = 0;
        for j = 1:n, c = c + u(j,i)*u(j,i); end
        c = sqrt(c);
        % Take new step length as the length of the first vector
        if i == 1
            stsize = c;
            if stsize < crit, mi = mi + 1; break; end
        end
    end

```

```

if c < crit % Orthogonality is lost: reset vectors.
    for j = 1:n
        for k = 1:n, v(j,k) = 0; end
        v(j,j) = 1; break;
    end
end
for j = 1:n, v(j,i) = u(j,i)/c; end
end
end
xopt = xt; fopt = ft;
end

```

Example 10.18: Rosenbrock's Method

Find the minimum of $f(x) = x_1^2 + 2x_2^2 + 2x_1x_2$ using Rosenbrock's method:

$$f(x) = x_1^2 + 2x_2^2 + 2x_1x_2$$

Use $x^0 = [0.5, 1]$ and $\text{crit} = 1 \times 10^{-6}$.

Solution

```

>> f = @(x) x(1)^2 + 2*x(2)^2 + 2*x(1)*x(2); x0 = [0.5 1]; crit = 1e-6;
>> [xopt,fopt,istag] = rbopt(f,x0,crit)
Convergence achieved.
xopt =
    -0.3000    0.2000
fopt =
    0.0500
istag =
    6

```

10.4.4 NELDER AND MEAD'S SIMPLEX METHOD

In an n -dimensional space, $n + 1$ points form a simplex. For example, a tetrahedron forms a simplex in three-dimensional space. An initial simplex in n -dimensional space can be formed by choosing the origin as one corner and n points, each marked at a distance of s from the origin along the coordinate axes. A regular simplex is constructed in space by adding the coordinates of a starting point to each of the $n + 1$ points. For minimization, function values at the $n + 1$ corner points of the simplex are evaluated first. The point with the highest function value is replaced by a newly created point that can be obtained by the steps reflection, expansion, or contraction.

Let f_h , f_s , and f_l be the highest, second highest, and the least function values among the $n + 1$ corners of the simplex, respectively, and let x_h , x_s , and x_l be the corresponding coordinates. Then x_m , the average of n points excluding the highest point, can be calculated by

$$x_m = \frac{1}{n} \sum_{j=1}^{n+1} x_j \quad (j \neq h)$$

If x_m is the mean of k points and the $(k + 1)$ th point is denoted by x_{k+1} , the updated mean of the $k + 1$ points is given by

$$x_m = x_m + \frac{x_{k+1} - x_m}{k + 1}$$

The reflected point x_r is given by $x_r = x_m + \gamma(x_m - x_h)$ (γ : reflection parameter, $\gamma > 0$), and the expanded point x_e is given by $x_e = x_r + \beta(x_r - x_m)$ (β : expansion parameter, $\beta > 1$). If $f_r = f(x_r) > f_h$, the contracted point x_c is given by $x_c = x_m + c(x_h - x_m)$ (c : contraction parameter, $0 < c < 1$). If $f_j < f_r \leq f_h$ ($j \neq h$), the contracted point x_c is given by $x_c = x_m + c(x_r - x_m)$. The point x_j is updated by $x_j = x_l + \alpha(x_j - x_l)$ ²⁰.

The Nelder and Mead's simplex method can be summarized as follows²¹:

1. Choose a starting point x_1 and evaluate $f_1 = f(x_1)$. Set $k = 0$.
2. Generate a simplex by creating n new corner points and calculate the function values at these points. Set $k = k + 1$.
3. Determine the highest, second highest, and the least function values f_h , f_s , and f_l and the corresponding coordinates x_h , x_s , and x_l , respectively. If $|f_h - f_l| < \epsilon$, go to step 13.
4. Determine x_m , the average of n points excluding the highest point, and the reflected point x_r . Calculate $f_r = f(x_r)$. If $f_r \geq f_l$, go to step 7.
5. Evaluate x_e and $f_e = f(x_e)$. If $f_e \geq f_l$, go to step 11. Replace x_h and f_h by x_e and f_e , respectively, and go to step 2.
6. Replace x_h and f_h by x_r and f_r , respectively, and go to step 2.
7. If $f_r > f_h$, go to step 9.
8. Replace x_h and f_h by x_r and f_r , respectively.
9. If $f_r \leq f_s$, go to step 2.
10. Evaluate x_c and $f_c = f(x_c)$. If $f_c > f_h$, go to step 11. Replace x_h and f_h by x_c and f_c , respectively, and go to step 2.
11. Perform update to find x_j and $f_j = f(x_j)$.
12. Calculate the function values at the new corners of the simplex and go to step 2.
13. If $k > 1$ and $|f_{\min} - f_l| < \epsilon$, the convergence is achieved and stop the calculation procedure.
14. Set $f_{\min} = f_l$ and replace x_1 and f_1 by x_l and f_l , respectively. Reduce the step reduction parameter and go to step 2.

The MATLAB function *nmopt* implements the Nelder and Mead's simplex method. The basic syntax is

```
[xopt, fopt, iter] = nmopt(fun, x0, crit)
```

where *fun* is the objective function, *x0* is the starting point vector, *crit* is the stopping criterion, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

```
function [xopt, fopt, iter] = nmopt(fun, x0, crit)
% nmopt.m: minimization by the Nelder and Mead's simplex method
% Inputs:
%   fun: objective functions
%   x0: starting point
%   crit: stopping criterion
% Outputs:
%   xopt: optimal point
%   fopt: objective function value at x=xopt
%   iter: number of iterations
% Example:
% x0 = [-3 -1 0 1]; crit = 1e-4;
% f = @(x) (x(1)+10*x(2))^2+5*(x(3)-x(4))^2+(x(2)-2*x(3))^4+10*(x(1)-x(4))^4;
% [xopt, fopt, iter] = nmopt(f, x0, crit)

rf = 1; % reflection parameter
ef = 2; % expansion parameter
```

```

cf = 0.5; % contraction parameter
sf = 0.5; % scale parameter
n = length(x0); beta = 1; n1 = n+1; iter = 0;
x = x0; fv(1) = fun(x);
for j = 1:n, pt(1,j) = x(j); end
% Create simplex and evaluate function values
for k = 2:n1
    u = zeros(1,n); u(1,k-1) = 1;
    pt(k,:) = pt(1,:) + beta*u; x = pt(k,:); fv(k) = fun(x);
end
while (1)
    nloop = 0;
    while (2)
        nloop = nloop + 1;
        if (nloop == 2), break; end
        iter = iter + 1;
        % Find highest, lease, and second highest values
        [fh, indh] = max(fv); [fl, indl] = min(fv); fs = fv(indl); inds = indl;
        for k = 1:n1
            if (k ~= indh)
                if (fs < fv(k)), fs = fv(k); inds = k; end
            end
        end
        % Find the average of n points xm
        for k = 1:n
            xm(k) = 0;
            for j = 1:n1
                if (j ~= indh), xm(k) = xm(k) + pt(j,k); end
            end
            xm(k) = xm(k)/n;
        end
        % Reflection procedure
        xr = xm + rf*(xm - pt(indh,:)); fr = fun(xr);
        if (fr >= fl & fr <= fs) % accept reflection
            fv(indh) = fr; pt(indh,:) = xr; break;
        end
        % Expansion procedure
        if (fr < fl )
            xe = xr + ef*(xr - xm); fe = fun(xe);
            if (fe < fl) % accept expansion
                fv(indh) = fe; pt(indh,:) = xe; break;
            else % accept reflection
                fv(indh) = fr; pt(indh,:) = xr; break;
            end
        end
        % Contraction procedure
        if (fr > fh)
            xc = xm + cf*(pt(indh,:) - xm); [fc] = fun(xc);
            if (fc <= fh) % accept contraction
                fv(indh) = fc; pt(indh,:) = xc; break;
            end
        elseif (fr > fs & fr <= fh)
            xc = xm + cf*(xr - xm); fc = fun(xc);
            if (fc <= fr) % accept contraction
                fv(indh) = fc; pt(indh,:) = xc; break;
            end
        end
    end
end

```

```

    end
    % Scaling
    for k = 1:n1
        if (k ~= indl)
            for j = 1:n, x(j) = sf *pt(k,j) + (1 - sf) *pt(indl,j);
            pt(k,j) = x(j); end
            fv(k) = fun(x);
        end
    end
    sigma = std(fv); avg = mean(fv);
    if (sigma <= crit)
        inc = inc + 1;
        if (inc == 2), break; end
    else
        inc = 0;
    end
end
xopt = pt(indl,:); fopt = fl;
end

```

Example 10.19: The Nelder and Mead's Simplex Method

Find the minimum of Powell's function using Nelder and Mead's simplex method:

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

As a starting point, use $x^0 = [-3, -1, 0, 1]$ and $\text{crit} = 1 \times 10^{-6}$.

Solution

```

>> x0 = [-3 -1 0 1]; crit = 1e-6;
>> f = @(x) (x(1)+10*x(2))^2+5*(x(3)-x(4))^2+(x(2)-2*x(3))^4+10*(x(1)-x(4))^4;
>> [xopt,fopt,iter] = nmopt(f,x0,crit)
xopt =
    -0.0244    0.0025   -0.0097   -0.0100
fopt =
    1.3627e-06
iter =
    80

```

10.4.5 SIMULATED ANNEALING (SA) METHOD

Metals have phase transformations when temperature changes. Annealing is a process where stresses are relieved from a previously hardened metal. More specifically, annealing is the physical process of heating up a metal above its melting point followed by cooling it down so slowly that the excited metal atoms can settle into a minimum energy level, yielding a crystal with a regular structure. Energy in the annealing process sometimes may increase even while the trend is a net decrease. This property can be applied to optimization problems.

Consider a simple minimization problem of the form

Minimize $f(x)$

Subject to $l_i \leq x_i \leq u_i$ ($i = 1, \dots, n$)

where $x = [x_1, x_2, \dots, x_n]^T$ is a column vector of n real-valued variables. In optimization applications, we begin with an initial state of temperature T that is set at a high level. The simulated annealing process can be implemented using Boltzmann's probability distribution function given by

$$p(\Delta E) = e^{-\frac{\Delta E}{kT}}$$

where k is Boltzmann's constant, which can be chosen as equal to 1 in optimization applications. The change Δf in the objective function value is accepted whenever it represents a decrease. If it is an increase, we accept it with a probability of $p(\Delta f) = e^{-\Delta f/T}$ ($k = 1$). The solution of minimization problem by the simulated annealing method can be summarized as follows²²:

1. Choose a starting point x and evaluate $f = f(x)$. Set $x_{\min} = x$ and $f_{\min} = f$.
2. Choose a vector s with step size s_i along the coordinate direction u_i . The initial value of each s_i may be set equal to a step size $s_T (=1)$ and a step reduction parameter r_s .
3. Define a vector a of acceptance ratios with each element equal to 1.
4. Choose a starting temperature T and a temperature reduction factor r_T . Set $k = 1$.
5. Set the temperature as $r_T T$ and the step size as $r_s s_T$. At each temperature, N_T iterations are carried out, and each iteration consists of N_C cycles.
6. Generate a random number r ($-1 \leq r \leq 1$) and evaluate a new point $x_s = x + rs_i u_i$. If this point lies in the outside of the bounds, the i th component of x_s is adjusted to be a random point in the interval l_i to u_i .
7. Evaluate $f_s = f(x_s)$. If $f_s \leq f$, the point is accepted by setting $x = x_s$. If $f_s \leq f_{\min}$, update f_{\min} and x_{\min} . If $f_s > f$, the point is accepted with a probability of $p = e^{-\frac{f-f_s}{T}}$.
8. Generate a random number r . If $r < p$, accept f_s . If a rejection takes place, update the acceptance ratio a_i as $a_i = a_i - \frac{1}{N_C}$.
9. Set $k = k + 1$ and go to step 5.
10. At the end of N_C cycles, use a_i to update the step size for the direction. A low value of a_i implies that there are more rejections suggesting that the step size should be reduced. A high rate of a_i implies more acceptances, which may be due to small step size. In this case, the step size is to be increased.

The MATLAB function *smopt* implements the simulated annealing method. The basic syntax is

```
[xopt, fopt, iter] = saopt(fun, T, xl, xu, rp, rs, crit)
```

where *fun* is the objective function, *T* is the initial temperature, *xl* and *xu* are the lower and upper bounds, *rp* is the temperature reduction factor, *rs* is the step reduction parameter, *crit* is the stopping criterion, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of cycle iterations.

```
function [xopt, fopt, iter] = saopt(fun, T, xl, xu, rp, rs, crit)
% saopt.m: minimization by simulated annealing (SA) method
% Inputs:
%   fun: objective function
%   T: initial temperature
%   xl,xu: lower and upper bounds on x
%   rp: temperature reduction factor
%   rs: step reduction parameter
%   crit: stopping criterion
% Outputs:
%   xopt: optimal point
%   fopt: objective function value at x=xopt
%   iter: number of cycle iterations
```

```

% Example:
%   f = @(x) -cos(5*sqrt(sum(x-5).^2)) + 0.1*sum(x-5).^2;
%   T = 100; xl = -10*[1 1]; xu = 10*[1 1]; rp = 0.8; rs = 0.9; crit = 1e-8;
%   [xopt,fopt,iter] = saopt(f,T,xl,xu,rp,rs,crit)

% Initialization
sf = 2; stp = 1; np = 10; nc = 20; nt = 2e4; ir = 16; rcrit = 1e-10; n =
length(xu);
% Feasible starting point
for j = 1:n, x(j) = xl(j) + rand*(xu(j) - xl(j)); xs(j) = x(j); xmin(j) =
x(j); end
f = fun(x); fmin = f; fold = f; citer = 0;
% Set step sizes, step factors and acceptance ratios
for j = 1:n, stsize(j) = stp; ar(j) = 1; end
while (1)
    for piter = 1:np % temperature step loop
        for ic = 1:nc % % search cycles: search along coordinate
            direction
            for k = 1:n
                xs(k) = x(k) + (2*rand - 1)*stsize(k);
                if (xs(k) < xl(k)) | (xs(k) > xu(k)), xs(k) = xl(k) +
                    rand*(xu(k) - xl(k));
                end
                fs = fun(xs);
                if fs <= f % point is accepted: update xmin and fmin
                    x(k) = xs(k); f = fs;
                    if fs < fmin, xmin = xs; fmin = fs; end
                    else
                        p = exp((f - fs)/T);
                        if rand < p, x(k) = xs(k); f = fs;
                        else % point rejected
                            xs(k) = x(k); ar(k) = ar(k) - 1/nc;
                        end
                    end
                end
            end
        end
    end
    % Adjust step so that about half the points are accepted.
    for j = 1:n
        if ar(j) > 0.6, stsize(j) = stsize(j)*(1 + sf*(ar(j) - 0.6)/0.4);
        elseif ar(j) < 0.4, stsize(j) = stsize(j)/(1 + sf*(0.4
            - ar(j))/0.4);
        end
        if stsize(j) > xu(j) - xl(j), stsize(j) = xu(j) - xl(j); end
        ar(j) = 1;
    end
    end
    stsize = stp*(xu - xl); fcrit = crit + rcrit*abs(fmin);
    if (fmin <= fold) & (fold - fmin < fcrit)
        citer = citer + 1;
        if citer >= ir, break; end
    else, citer = 0;
    end
    % Reduce temperature
    T = rp*T; stp = rs*stp; x = xmin; f = fmin; fold = f;
end
xopt = xmin; fopt = fmin; iter = citer;
end

```

Example 10.20: The Simulated Annealing Method

Find the minimum of the corrugated spring function $f(x)$ using the simulated annealing method²³:

$$f(x) = -\cos(kR) + 0.1R^2, \quad R = \sqrt{(x_1 - c)^2 + (x_2 - c)^2}, \quad -10 \leq x_1, x_2 \leq 10$$

Set the initial temperature as $T = 100$, the temperature reduction factor $rp = 0.8$, the step reduction parameter $rs = 0.9$, and $crit = 1 \times 10^{-8}$.

Solution

The corrugated spring function is defined by the function fcy as follows:

```
function fv = fcy(x)
% Corrugated spring function
C = 0; for j = 1:2, C = C + (x(j) - 5)^2; end
fv = -cos(5*sqrt(C)) + 0.1*C;
end
```

The following commands carry out the SA calculation procedure:

```
>> T = 100; xl = -10*[1 1]; xu = 10*[1 1]; rp = 0.8; rs = 0.9; crit = 1e-8;
>> [xopt,fopt,iter] = saopt(@fcy,T,xl,xu,rp,rs,crit)
xopt =
    5.0000    5.0000
fopt =
    -1.0000
iter =
    16
```

10.4.6 GENETIC ALGORITHM (GA)

The natural process takes place through the mutation and recombination of the chromosomes in the population to yield a better gene structure. The genetic algorithm is a typical direct search method modeled on the natural evolution and selection processes toward the survival of the fittest. One of the significant characteristics of the genetic algorithm is the use of stochastic information in its implementation, and therefore this method may be considered as a global optimization method. A basic genetic algorithm can be summarized as follows²⁴:

1. Set a set of solution or chromosomes (population) $P(0)$. Evaluate the initial solution for fitness. Set $t = 0$.
2. Generate the set of children (crossover, mutation) using the genetic operators. Add a new set of randomly generated population and evaluate again the population fitness.
3. Determine which members will be part of next generation (competitive selection). Select population $P(t + 1)$.
4. If the convergence is not achieved, set $t = t + 1$ and go to step 2.

Figure 10.5 shows the flowchart for a genetic algorithm.

In general, the problem for the genetic algorithm is posed in the form

Maximize $f(x)$

Subject to $l_i \leq x_i \leq u_i$ ($i = 1, \dots, n$)

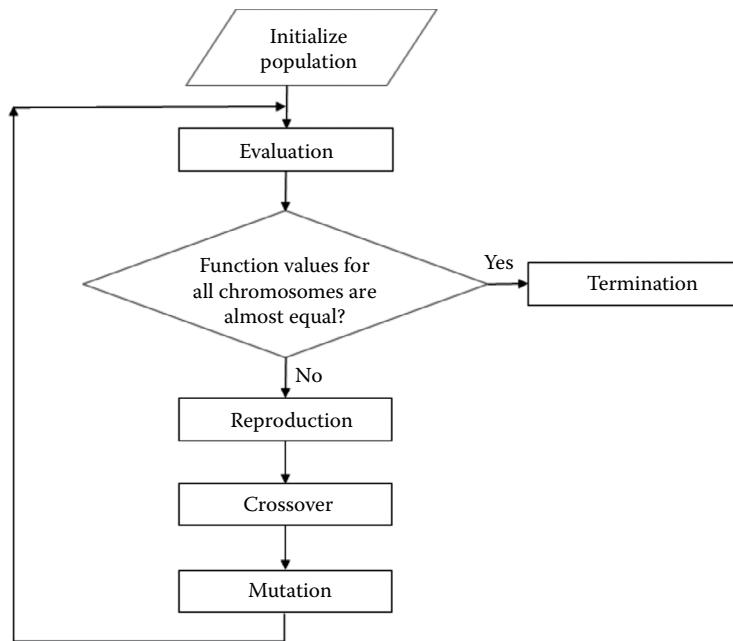


FIGURE 10.5 Flowchart for a genetic algorithm. (From Yang, W.Y. et al., *Applied Numerical Methods Using MATLAB*, Wiley Interscience, Hoboken, NJ, 2005, p. 338.)

where $x = [x_1, x_2, \dots, x_n]^T$ is a column vector of n real-valued variables. Each variable is represented as a binary number, say, of m bits. This is performed by dividing the feasible interval of variable x_i into $2^m - 1$ intervals. The interval step s_i for variable x_i is given by

$$s_i = \frac{u_i - l_i}{2^m - 1}$$

In the creation of initial population, each member of the population is a string of size $n * m$ bits. In the evaluation phase, the values of variables for each member are extracted. The function values f_1, f_2, \dots are evaluated. These values are referred to as fitness values. The reproduction takes the steps of creation of a mating pool, crossover, and mutation operations. The sum of the scaled fitness value S is calculated as

$$S = \sum_{j=1}^z f'_j, \quad f'_j = \frac{f_j + Q}{R}, \quad Q = 0.1f_h - 1.1f_l, \quad R = \max(1, f_h + Q)$$

Roulette wheel selection is used to make copies of the fittest members for reproduction. The crossover operation is carried out on the mating parent pool to produce offspring. There is a chance that the parent selection and crossover operations may result in close to identical individuals that may not be the fittest. Mutation is performed in which random operation is carried out to change the expected fitness. The population is evaluated again, and the highest fitness value and the corresponding variable values are stored as f_{\max} and x_{\max} . This completes a generation. If the predetermined number of generations is complete, the calculation process is terminated.

The MATLAB function `gaopt` implements the genetic algorithm. The basic syntax is

```
[xopt, fopt, iter] = gaopt(fun, xl, xu, nb, ps, ng, mp)
```

where *fun* is the objective function, *xl* and *xu* are the lower and upper bounds, *nb* is the number of binary digits, *ps* is the population size, *ng* is the number of generations, *mp* is the mutation probability, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of function evaluations.

```

function [xopt,fopt,iter] = gaopt(fun,xl,xu,nb,ps,ng,mp)
% gaopt.m: maximization by the genetic algorithm
% Inputs:
%   fun: objective functions
%   xl,xu: lower and upper bounds on x
%   nb: number of binary digits
%   ps: population size
%   ng: number of generations
%   mp: mutation probability
% Outputs:
%   xopt: optimal point
%   fopt: objective function value at x=xopt
%   iter: number of function evaluations
% Example:
%   xl=-6*[1 1 1 1]; xu=6*[1 1 1 1]; nb=8; ps=40; ng=50; mp=0.02;
%   f = @(x) (x(1)+10*x(2))^2+5*(x(3)-x(4))^2+(x(2)-2*x(3))^4+10*(x(1)-x(4))^4;
%   [xopt,fopt,iter] = gaopt(f,xl,xu,nb,ps,ng,mp)

n = length(xu);
Pn = round(rand(ps,n*nb)); % initial population
for kg = 1:ng
    ft = fitness(fun,ps,nb,n,xl,xu,Pn); [fmax,kmax] = max(ft);
    if (kg == 1)
        fbest = fmax;
        for k = 1:n
            ab = num2str(Pn(kmax,nb*(k-1)+1:nb*k));
            adec = bin2dec(ab); % convert binary string to decimal integer
            xbest(k) = xl(k) + (xu(k)-xl(k))/(2^nb - 1)*adec;
        end
    else
        if (fmax > fbest)
            fbest = fmax;
            for k = 1:n
                ab = num2str(Pn(kmax,nb*(k-1)+1:nb*k));
                adec = bin2dec(ab);
                xbest(k) = xl(k) + (xu(k)-xl(k))/(2^nb - 1)*adec;
            end
        end
    end

    [frk,ft] = frank(ps, ft); % scaling
    for k = 1: ps % shuffling: roulette wheel
        ik = frk(k); ptemp = ps; ktemp = k; am(ik) = 2*(ptemp + 1 -
            ktemp)/(ptemp*(ptemp + 1));
    end
    for k = 2: ps, ptemp = am(k - 1); am(k) = ptemp + am(k); end
    %
    for k = 1:ps % selection
        kstr1 = roul(ps,am); kstr2 = roul(ps,am); kchd =
            cros(Pn,nb,n,kstr1, kstr2);
    end
end

```

```

kchd = mutat(kchd,mp,nb,n);
for j = 1: nb*n, Pn(k, j) = kchd(j); end
end
xopt = xbest; fopt = fbest; iter = ng*ps;
end

function ft = fitness(fun,ps,nb,n,xl,xu,Pn) % function evaluation
for k = 1:ps
    for j = 1:n
        a = num2str(Pn(k,nb*(j-1) + 1:nb*j)); adec = bin2dec(a);
        x(j) = xl(j) + (xu(j)-xl(j))/(2^nb - 1)*adec;
    end
    f = fun(x); ft(k) = f;
end
end

function [frk,ft] = frank(ps, ft)
for k = 1:ps, frk(k) = k; end
for k = 1:ps-1
    temp = ft(k); ktemp = k;
    for j = k+1:ps
        if (ft(j) > temp), temp = ft(j); ktemp = j; end
    end
    jtemp = frk(ktemp); frk(ktemp) = frk(k); frk(k) = jtemp;
    ftemp = ft(ktemp); ft(ktemp) = ft(k); ft(k) = ftemp;
end
end

function kstr = roul(ps,am) % shuffling operation
temp = rand;
for k = 1:ps
    while (1)
        if (temp > am(k)), break; end
        kstr = k; return;
    end
end
end

function kchd = cros(Pn,nb,n,kstr1, kstr2) % crossover operation
for j = 1:n
    jp = nb*(j - 1); nc = floor(rand*nb + 0.5);
    if (nc == 0)
        for k = 1:nb, kchd(k+jp) = Pn(kstr2, k+jp); end
    elseif (nc == nb)
        for k = 1:nb, kchd(k+jp) = Pn(kstr1, k+jp); end
    else
        for k = 1: nc, kchd(k+jp) = Pn(kstr1, k+jp); end
        for k = nc+1:nb, kchd(k+jp) = Pn(kstr2, k+jp); end
    end
end
end

function [kchd] = mutat(kchd,mp,nb,n) % mutation operation
for k = 1:nb*n

```

```

if (rand <= mp)
    if (kchd(k) == 1), kchd(k) = 0;
    else, kchd(k) = 1;
    end
end
end

```

Example 10.21: The Genetic Algorithm

Find the minimum of the corrugated spring function $f(x)$ using the genetic algorithm²⁶:

$$f(x) = -\cos(kR) + 0.1R^2, \quad R = \sqrt{(x_1 - c)^2 + (x_2 - c)^2}, \quad -10 \leq x_1, x_2 \leq 10$$

Set nb = 8, ps = 50, ng = 60, and mp = 0.05.

Solution

Note that the function *gaopt* carries out the maximization problem. Thus, the corrugated spring function defined in the previous example (function *fcy*) should be multiplied by -1 as follows:

```

function fv = fcy2(x)
% Corrugated spring function
C = 0; for j = 1:2, C = C + (x(j) - 5)^2; end
fv = -(-cos(5*sqrt(C)) + 0.1*C);
end

```

The following commands carry out the GA calculation procedure:

```

>> xl = -10*[1 1]; xu = 10*[1 1]; nb = 8; ps = 50; ng = 60; mp = 0.05;
>> [xopt,fopt,iter] = gaopt(@fcy2,xl,xu,nb,ps,ng,mp)
xopt =
    4.9020    4.9804
fopt =
    0.8766
iter =
    3000

```

10.5 MIXED-INTEGER PROGRAMMING

10.5.1 ZERO-ONE PROGRAMMING METHOD

Many engineering problems can be formulated as zero-one problems. For example, selection of optimal locations of actuators may be formulated as a zero-one problem. Moreover, a certain class of integer programming problems may be converted into equivalent zero-one linear programming problems. Consider a simple minimization problem of the form

Minimize $f(x)$
Subject to $g_i(x) \leq b_i, x_i = 0 \text{ or } 1 (i = 1, \dots, n)$

where $x = [x_1, x_2, \dots, x_n]^T$ is a vector of n real-valued variables. If there are inequality constraints of “greater than” (\geq) type, these can be converted into constraints of “less than” (\leq) type. The search for the solution starts with an initial point $x = 0$. If this initial solution is also feasible, then we are done. If it is not feasible, the variable values are adjusted so that feasibility is attained.

The MATLAB function *ozopt* implements the zero–one programming method. The basic syntax is

```
[xopt,fopt,iter] = ozopt(A,c,nl,ne)
```

where *A* is the coefficient matrix for constraints, *c* is a row vector of coefficients of the objective function, *nl* is the number of “less than” (\leq) type constraints, *ne* is the number of equality ($=$) constraints, *nint* is the number of integer variables, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations. The last column of the matrix *A* contains the right-hand sides of “less than” inequality constraints.

```
function [xopt,fopt,iter] = ozopt(A,c,nl,ne)
% ozopt.m: zero-one programming method for integer minimization problem
%           inequality constraints should be rearranged to <= constraints.
% Inputs:
%   A: coefficient matrix for constraints including right hand sides
%   c: coefficient vector for objective function (row vector)
%   nl: number of <= constraints
%   ne: number of = constraints
% Outputs:
%   xopt: optimal point (0 or 1)
%   fopt: function value at the optimal point (=f(xopt))
%   iter: number of iterations
% Example
%   nl = 3; ne = 0;
%   A = [-1 3 6 6; -2 -3 -3 -2; -1 0 -1 -1]; c = [4 5 3];
%   [xopt,fopt,iter] = ozopt(A,c,nl,ne)

% nl = 3; ne = 3;
% A = [1 0 0 1 0 1 0 0 0 1; 0 1 0 0 0 0 1 0 0 1; 0 0 0 1 0 0 0 0 0 1 1 1
%      0 0 0 0 0 1; 0 0 0 1 1 0 0 0 0 1; 0 0 0 0 0 0 0 1 1 1];
% c = -[5 3 1 3 5 2 5 5 2];

nv = length(c); m = nl + ne;
x = zeros(1,nv); Am = zeros(m+1,nv+1); xvec = zeros(1,nv);
indv = zeros(1,nv); temA = zeros(1,m+1); xmin = zeros(1,nv);
[rAm,cAm] = size(A); Am(1:rAm,1:cAm) = A;
for k = 1:m, Am(k,nv+1) = -Am(k,nv + 1); end
% Convert <= constraints to >=
for k = 1:nl
    for j = 1:nv+1, Am(k,j) = -Am(k,j); end
end
% Convert = constraints to >= by adding one >= constraint
if ne > 0
    m = m + 1;
    for k = nl+1:m-1
        for j = 1:nv+1, Am(m,j) = Am(m,j) - Am(k,j); end
    end
end
% Convert variables if c(j) < 0
check0 = 0;
for j = 1:nv
    if c(j) < 0
        indv(j) = 1; check0 = check0 + c(j); c(j) = -c(j);
    end
end
```

```

for k = 1:m
    Am(k,nv+1) = Am(k,nv+1) + Am(k,j); Am(k,j) = -Am(k,j);
end
end
for k = 1:nv, xvec(k) = k; x(k) = 0; end
for k = 1:m, temA(k) = Am(k,nv+1); end
indf = 1; inds = 0; f = 0; iter = 0;
while (1)
    sflag = 0; iter = iter + 1;
    if xvec(indf) > -1
        sflag = 1;
        for k = 1:m
            if temA(k) < 0, sflag = 0; break; end
        end
    end
    if sflag == 1 % feasible solution
        inds = inds + 1;
        if inds == 1 || (inds > 1 && f < fmin)
            fmin = f;
            for k = 1:nv, xmin(k) = x(k); end
        end
    end
    cflag = 0; nflag = 0;
    if sflag == 0 % infeasible solution
        for k = 1:m
            indk = temA(k);
            if indk < 0
                for j = indf+1:nv
                    if Am(k,xvec(j)) > 0, indk = indk + Am(k,xvec(j));
                end
            end
            if indk < 0, cflag = 1; break; end
        end
        if inds > 0
            nflag = 1;
            for k = indf+1:nv
                if f + c(xvec(k)) < fmin, nflag = 0; break; end
            end
        end
    end
    if sflag == 1 || cflag == 1 || nflag == 1
        while xvec(indf) < 0
            xvec(indf) = -xvec(indf); indf = indf - 1;
            if indf == 0, break; end
        end
        if indf == 0, break; end
        x(xvec(indf)) = 0;
        % Update constraints and function value (f)
        for k = 1:m, temA(k) = temA(k) - Am(k,xvec(indf)); end
        f = f - c(xvec(indf)); xvec(indf) = -xvec(indf);
    else
        indf = indf + 1;
        for k = indf:nv
            difa = 0;

```

```

for j = 1:m
    temg = temA(j) + Am(j,xvec(k));
    if temg < 0, difa = difa - temg; end
end
if k == indf
    delf = difa; ink = k;
else
    if delf > difa, delf = difa; ink = k; end
end
inj = xvec(indf); xvec(indf) = xvec(ink); xvec(ink) = inj;
% Update x and f
x(xvec(indf)) = 1;
for k = 1:m, temA(k) = temA(k) + Am(k,xvec(indf)); end
f = f + c(xvec(indf));
end
end
if inds == 0, display('No feasible solution.'); return; end
% Assign outputs
for k = 1:nv
    if (indv(k) == 1), xmin(k) = 1 - xmin(k); end
end
xopt = xmin; fopt = fmin;
end

```

Example 10.22: The Zero–One Programming Method

Find the solution of the following minimization problem using the zero–one programming method:

Minimize $f(x) = 4x_1 + 3x_2$
 Subject to $2x_1 - 5x_2 \leq 10$, $3x_1 + 2x_2 \leq 9$, $0 \leq x_1, x_2$ ($x_i = 0$ or 1)

Solution

```

>> nl = 2; ne = 0; A = [2 -5 10;3 2 9]; c = [4 3];
>> [xopt,fopt,iter] = ozopt(A,c, nl, ne)
xopt =
     0      1
fopt =
     -1
iter =
     4

```

10.5.2 BRANCH AND BOUND METHOD

The branch and bound method is useful to solve mixed-integer linear programming problems. In this method, a continuous relaxed linear programming problem is solved at every solution stage. If the solution of the continuous problem happens to be an integer solution, it represents the optimum solution of the integer problem. Otherwise, at least one of the integer variables must assume a noninteger value. If a variable x_i is not an integer, it can be represented by

$$I_x < x_i < I_x + 1$$

where I_x is the largest integer bounded by x_i . For example, if $x_i = 4.7$, $I_x = 4$. In this case, two subproblems are formulated, one with the additional upper bound constraint and another with the lower bound constraint as follows:

$$x_i \leq I_x, \quad x_i \geq I_x + 1$$

In this branching process, some portion of the continuous space that is not feasible for the integer problem is eliminated while all the integer feasible solutions are retained. The solution of a continuous problem forms a node, and the linear programming problem is solved at each node k . The process of branching and solving a sequence of continuous problems is continued until an integer feasible solution is found for one of the two continuous problems.²⁷ The branch and bound algorithm can be summarized as follows²⁸:

1. Set the root node with all discrete variables free. Set the current incumbent solution to $f^* = \infty, x^* = [\infty]$.
2. If any active feasible partial solutions remain, choose one as x^s . If none exists and there exists an incumbent solution, it is done. If none exists and there is no incumbent solution, the problem is infeasible.
3. Obtain the completion of the partial solution x^s using a continuous relaxation of the original problem.
4. If there are no feasible completions of the partial solution x^s , set $s + 1 \rightarrow s$ and go to step 2.
5. If the best feasible completion of the partial solution x^s cannot improve the incumbent solution, set $s + 1 \rightarrow s$ and go to step 2.
6. If the best feasible completion is better than the incumbent, update the incumbent solution as $f^s \rightarrow f^*$ and $x^s \rightarrow x^*$. Set $s + 1 \rightarrow s$ and go to step 2.

The MATLAB function *bnbopt* implements the branch and bound algorithm. The basic syntax is

```
[xopt, fopt, iter] = bnbopt(A, b, c, nl, ng, ne, ibd)
```

where *A* is the coefficient matrix for constraints, *b* is a row vector representing the right-hand side of constraints, *c* is a row vector of coefficients of the objective function, *nl* is the number of “less than” (\leq) type constraints, *ng* is the number of “greater than” (\geq) type constraints, *ne* is the number of equality ($=$) constraints, *ibd* is the variable index vector, *xopt* is the resultant optimum point, *fopt* is the function value at the optimum point, and *iter* is the number of iterations.

```
function [xopt, fopt, iter] = bnbopt(A, b, c, nl, ng, ne, ibd)
% bnbopt.m: branch and bound method for mixed inter minimization problem
% Inputs:
%   A: coefficient matrix for constraints
%   b: right hand side of constraints (row vector)
%   c: coefficient vector for objective function (row vector)
%   nl: number of <= constraints
%   ng: number of >= constraints
%   ne: number of = constraints
%   ibd: variable index vector
% Outputs:
%   xopt: optimal point (0 or 1)
%   fopt: function value at the optimal point (-f(xopt))
%   iter: number of iterations

critd = 1e-4; critf = 5e-2; kmax = 1e4; ksol = 0; kf = 0; fmin = 1e30;
nv = length(c); nbd = length(ibd);
for j = 1:length(b)
    if (b(j) < 0), disp('b(j) must be non-negative.'), return; end
end
for k = 1:nv, vtype(k) = 1; end
for k = 1:nbd, vtype(ibd(k)) = 2; end
for j = 1:nv, kvar(j) = j; end
iter = 0; kconv = 0;
```

```
while (1)
    iter = iter + 1;
    if (iter > kmax)
        disp('Maximum possible iterations exceeded'); break;
    end
    ktrac = 0; [f,x,kflag] = linpm(nv,nl,ng,ne,kf,kvar,A,b,c);
    if (nbd == 0)
        if (kflag > 0), disp('No feasible solution.'), return;
        else
            fmin = f; xmin = x; break;
        end
    end
    jc = 0;
    while (2)
        jc = jc+1; if (jc == 2), break; end
        if (kflag > 0)
            if (kf == 0), disp('No feasible solution.'), return; end
            ktrac = 1; break; % backtrack operation
        else
            if (iter == 1), flb = f; end % best low bound
            if (f >= fmin), ktrac = 1; break; end
            vmax = 0;
            for j = 1:nv
                if(vtype(j) == 2)
                    diffx = abs(x(j)-round(x(j)));
                    if (diffx > vmax), vmax = diffx; jv = j; end
                end
            end
            if (vmax < critd)
                ksol = 1;
                if (f < fmin)
                    fmin = f; for j = 1:nv, xmin(j) = x(j); end
                end
                gapf = abs(flb - fmin)/abs(flb);
                if (gapf <= critf), kconv = 1; break; end
                ktrac = 1; % backtrack operation
                break
            else
                kf = kf + 1; jt = kvar(kf); kvar(kf) = kvar(jv); kvar(jv) = jt;
            end
        end
    end
    while (ktrac == 1)
        if (kvar(kf) > 0), kvar(kf) = -kvar(kf); break;
        else
            kvar(kf) = -kvar(kf); kf = kf - 1;
            if (kf == 0)
                if(ksol == 0), disp('No feasible solution.'), return; end
                kconv = 1; break;
            end
        end
    end
    if (kconv == 1), break;
end
xopt = xmin; fopt = fmin;
end
```

```

function [f,x,kflag] = linpm(nv,nl,ng,ne,kf,kvar,A,B,C)
if (kf > 0)
    idn = [1:nv];
    for k = 1:kf, jk = abs(kvar(k)); idn(jk) = 0; end
    difnv = nv - kf; nej = ne; Aj = A; Bj = B; f0 = 0; cj = 0; jm = 0;
    for k = 1:nv
        if idn(k) > 0, cj = cj+1; Ck(cj) = C(k);
        else
            jm = jm+1;
            if (kvar(jm) > 0), cm = 1; else, cm = 0;
            end
            f0 = f0 + C(k)*cm;
        end
    end
    for k = 1:nl, conk(k) = -1; end
    for k = 1:ng, conk(nl+k) = 1; end
    for k = 1:nl+ng+ne
        cj = 0; jm = 0;
        for j = 1:nv
            if (idn(j)) > 0, cj = cj + 1; Aj(k,cj) = A(k,j);
            else
                jm = jm+1;
                if (kvar(jm) > 0), cm = 1; else, cm = 0; end
                Bj(k) = Bj(k) - A(k,j)*cm;
            end
        end
        if (k <= nl+ng & Bj(k) < 0)
            conk(k) = -conk(k); Aj(k,:) = -Aj(k,:); Bj(k) = -Bj(k);
        end
    end
    nlk = 0; ngk = 0;
    for k = 1:nl+ng
        if (conk(k) == -1), nlk = nlk + 1;
        elseif (conk(k)==1), ngk = ngk + 1;
    end
    end
    [ids,xk] = sort(conk); A = Aj; B = Bj;
    for k = 1:nl+ng, A(k,:) = Aj(xk(k),:); B(k) = Bj(xk(k)); end
    nk = nlk + ngk + nej; Aj = A(1:nk,1:difnv); Bj = B(1:nk);
else
    difnv = nv; nlk = nl; ngk = ng; nej = ne; Aj = A; Bj = B; Ck = C;
end
[fk,xs,kflag] = linsimx(difnv,nlk,ngk,nej,Aj,Bj,Ck);
if (kflag > 0), x = xs; f = fk; return; end
if (kf > 0)
    cj = 0; jm = 0;
    for j = 1:nv
        if (idn(j)) > 0. cj = cj+1; x(j) = xs(cj);
        else
            jm = jm+1;
            if (kvar(jm)>0), cm = 1; else, cm = 0; end
            x(j) = cm;
        end
    end
    f = fk + f0;
else

```

```

f = fk; x = xs; return;
end
end

function [f,x,kflag] = linsimx(nv,nl,ng,ne,A,B,C)
% linear programming by simplex method
mnp = 20; bigm = 0; kflag = 0;
nc = nv + nl + ne + 2*ng; nr = nl + ne + ng + 1;
% Initialization
Aj(1:nr,1:nc) = 0; Bj(1:nr) = 0; Aj(1:nr-1,1:nv) = A(1:nr-1,1:nv);
Aj(nr,1:nv) = C(1:nv); Bj(1:nr-1) = B(1:nr-1); A = Aj; B = Bj;
for j = 1:nv, A(nr,j) = C(j); end
for j = 1:nv, bigm = bigm + mnp*abs(A(nr,j)); end
if (nl > 0) % slack variables
    for k = 1:nl, A(k,nv+k) = 1; Bs(k) = nv + k; end
end
if (ng > 0)
    for k = 1:ng
        A(nl+k,nv+nl+k) = -1; A(nl+k, nv+nl+ng+k) = 1;
        Bs(nl+k) = nv + nl + ng + k;
    end
end
if (ne > 0)
    for k = 1:ne
        A(nl+ng+k,nv+nl+2*ng+k) = 1; Bs(nl+ng+k) = nv + nl + 2*ng + k;
    end
end
mge = ng + ne;
if (mge > 0), for k = 1:mge, A(nr,nv+nl+ng+k) = bigm; end
end
if (mge > 0) % Remove artificial variables
    for k = 1:mge
        C = A(nr,nv+nl+ng+k);
        for j = 1:nc, A(nr,j) = A(nr,j) - C*A(nl+k,j); end
        B(nr) = B(nr) - C*B(nl+k);
    end
end
% Simplex method
while (1)
    jflag = 0;
    for j = 1:nc
        if (A(nr,j) < 0), jflag = 1; break; end
    end
    if (jflag == 0), break; end
    C = bigm;
    for j = 1:nc
        if (A(nr,j) < C), C = A(nr,j); idv = j; end
    end
    jn = 0; jk = 0;
    for k = 1:nr-1
        if (A(k,idv) > 0)
            jk = jk + 1; Dm = B(k)/(A(k,idv) + 1e-10);
            if (jk == 1), C = Dm; jp = k;
            else, if (Dm < C), C = Dm; jp = k; end
            end
            jn = 1;
        end
    end

```

```

end
Bs(jp) = idv; if (jn == 0), kflag = 1; f = 0; x = 0; return; end
Dm = 1/A(jp,idv); B(jp) = Dm*B(jp);
for j = 1:nc, A(jp,j) = Dm*A(jp,j); end
for k = 1:nr
    if (k ~= jp)
        Em = A(k,idv); for j = 1:nc, A(k,j) = A(k,j) - Em*A(jp,j); end
        B(k) = B(k) - Em*B(jp);
    end
end
x(1:nv) = 0; nt = nv + nl + ng;
for k = 1:nr-1
    if (Bs(k) > nt), kflag = 1; f = 0; x = 0; return; end
end
for k = 1:nv
    for j = 1:nr-1
        if (Bs(j) == k), x(k) = B(j); break; end
    end
end
f = B(nr); f = -f; % minimization
end

```

Example 10.23: The Branch and Bound Algorithm (1)

Find the solution of the following minimization problem using the branch and bound algorithm:

$$\begin{aligned}
 & \text{Minimize } f(x) = 12x_1 + 10x_2 + 6x_3 + 4x_4 \\
 & \text{Subject to } x_1 + 3x_2 + 2x_3 + 4x_4 \leq 8, \quad x_1 + x_2 + x_3 = 2, \quad x_3 + x_4 = 1 \quad (x_i = 0 \text{ or } 1)
 \end{aligned}$$

Solution

```

>> A = [1 3 2 4;1 1 1 0;0 0 1 1]; b = [8 2 1]; c = [12 10 6 4]; nl = 1; ne =
2; ng = 0; ibd = [1 2 3 4];
>> [xopt,fopt,iter] = bnbopt(A,b,c, nl, ng, ne, ibd)
xopt =
0 1 1 0
fopt =
16
iter =
1

```

Example 10.24: The Branch and Bound Algorithm (2)

Find the solution of the following maximization problem using the branch and bound algorithm²⁹:

$$\begin{aligned} & \text{Maximize } f(x) = 5x_1 + 3x_2 + x_3 + 3x_4 + 5x_5 + 2x_6 + 5x_7 + 5x_8 + 2x_9 \\ & \text{Subject to } x_1 + x_4 + x_6 \leq 1, \quad x_2 + x_7 \leq 1, \quad x_4 + x_9 \leq 1, \end{aligned}$$

$$x_1 + x_2 + x_3 = 1, \quad x_4 + x_5 = 1, \quad x_8 + x_9 = 1, \quad (x_i = 0 \text{ or } 1)$$

Solution

```

>> A = [1 0 0 1 0 1 0 0 0;0 1 0 0 0 0 1 0 0;0 0 0 1 0 0 0 0 0 1;1 1 1 0 0 0 0 0 0
0;0 0 0 1 1 0 0 0 0;0 0 0 0 0 0 0 0 1 1];
>> b = ones(1,6); c = -[5 3 1 3 5 2 5 5 2]; nl = 3; ne = 3; ng = 0; ibd = [1:9];
>> [xopt,fopt,iter] = bnbopt(A,b,c, nl, ng, ne, ibd)

```

```

xopt =
    1      0      0      0      1      0      1      1      0
fopt =
    -20
iter =
    1

```

We can see that the maximum function value is $f_{\max} = 20$.

10.6 USE OF MATLAB BUILT-IN FUNCTIONS

MATLAB provides many useful functions that allow solving various optimization problems. Table 10.1 lists the names of MATLAB built-in optimization routines.

10.6.1 UNCONSTRAINED OPTIMIZATION

10.6.1.1 fminsearch

The function *fminsearch* can be used to solve a multidimensional unconstrained nonlinear minimization problem. This function uses the Nelder–Mead simplex (direct search) method. The basic syntax is

```
[x, f] = fminsearch(fun, x0)
```

where f is the function value at the optimum point, x_0 is the initial search point, fun is a function handle, and x is a local minimizer of fun .

10.6.1.2 fminunc

The function *fminunc* finds a local minimum of a function of several variables. This function can be used as

```
[x, f] = fminunc(fun, x0)
```

where f is the function value at the optimum point, x_0 is the initial search point, fun is a function handle, and x is a local minimizer of fun .

TABLE 10.1
MATLAB Built-In Optimization Routines

Problem Types	Function Name	Description
Unconstrained optimization	fminbnd	Bounded nonlinear minimization
	fminsearch	Unconstrained nonlinear minimization
	fminuc	Local minimum of a function of several variables
	lsqnonlin	Nonlinear least-squares problem
Constrained optimization	lsqlin	Constrained linear least-squares problem
	fmincon	Constrained minimum of a function
	fminimax	Minimax solution of a function
	quadprog	Quadratic programming problem
Linear programming	linprog	Linear programming problem
Mixed-integer programming	intlinprog	Mixed-integer linear programming problem

10.6.1.3 fminbnd

The function *fminbnd* solves a single-variable bounded nonlinear function minimization problem. If we want to find a local minimum of the function *fun* in the interval $x_1 < x < x_2$, we can execute the following commands:

```
[x, f] = fminbnd(fun, x1, x2)
```

where *x1* and *x2* denote the lower and upper bounds, *fun* is a function handle, and *x* is a local minimizer of *fun*.

10.6.1.4 lsqnonlin

The function *lsqnonlin* attempts to solve nonlinear least-squares problems of the form

$$\text{Minimize } \sum f(x)^2$$

The basic syntax is

```
x = lsqnonlin(fun, x0, lb, ub)
```

where *x0* is the starting point, *fun* is a function handle, *lb* and *ub* are the lower and upper bounds ($lb < x < ub$), and *x* is a local minimizer of *fun*. If no bounds exist, we may use empty matrices for *lb* and *ub* or just omit them. If x_i is unbounded below, set *lb(i)* = $-\text{Inf}$, and set *ub(i)* = Inf if x_i is unbounded above.

Example 10.25: Unconstrained Optimization by Built-In Functions

1. Find the minimum of $f(x) = 2x^2 \sin(x) + e^{-x}$ using the built-in functions *fminsearch* and *fminunc*.
2. Minimize $f(x) = 2x^2 \sin(x) + e^{-x}$ using the built-in function *fminbnd* in the interval $-4 \leq x \leq 0$.
3. Minimize $f(x) = 2x^2 \sin(x) + e^{-x}$ using the built-in function *lsqnonlin* in the interval $-4 \leq x \leq 0$. Use $x_0 = -1$ as a starting point. Check the result using the function *fminbnd*.

Solution

1. Let the starting point be $x_0 = 1$:

```
>> f = @(x) 2*x^2*sin(x) + exp(-x); x0 = 1;
>> [x, fv] = fminsearch(f, x0)
x =
0.3488
fv =
0.7887
>> [x, fv] = fminunc(f, x0)
x =
0.3488
fv =
0.7887
```

2. For the interval $-4 \leq x \leq 0$, $x_1 = \text{lb} = -4$ and $x_2 = \text{ub} = 0$.

```
>> f = @(x) 2*x^2*sin(x) + exp(-x); lb = -4; ub = 0;
>> [x, fv] = fminbnd(f, lb, ub)
x =
-1.7540
fv =
-0.2724
```

3. For the interval $-4 \leq x \leq 0$, $x_1 = \text{lb} = -4$ and $x_2 = \text{ub} = 0$, and $x_0 = -1$.

```
>> f = @(x) 2*x^2*sin(x) + exp(-x); lb = -4; ub = 0; x0 = -1;
>> x = lsqnonlin(f, x0, lb, ub)
```

```

x =
-1.4962
>> f = @(x) (2*x^2*sin(x) + exp(-x))^2; lb = -4; ub = 0; x0 = -1;
>> [x,fv] = fminbnd(f, lb ,ub)
x =
-1.4962
fv =
6.5941e-11

```

10.6.2 CONSTRAINED OPTIMIZATION

10.6.2.1 lsqlin

The function *lsqlin* solves constrained linear least-squares problems of the form

$$\begin{aligned} & \text{Minimize } \frac{1}{2} \|Cx - d\|^2 \\ & \text{Subject to } Ax \leq b, A_{eq}x = b_{eq} \end{aligned}$$

The basic syntax is

```
x = lsqlin(C, d, A, b, Aeq, beq, lb, ub, x0)
```

where $C \in R^{m \times n}$ is a coefficient matrix, x_0 is the starting point, fun is a function handle, lb and ub are the lower and upper bounds ($lb < x < ub$), and x is a local minimizer of fun . If no bounds exist, we may use empty matrices for lb and ub or just omit them. If x_i is unbounded below, set $lb(i) = -\text{Inf}$, and set $ub(i) = \text{Inf}$ if x_i is unbounded above. Depending on the problem type or conditions, the corresponding input arguments may be omitted. For example, if there exist only inequality constraints, we can use

```
x = lsqlin(C, d, A, b)
```

10.6.2.2 fmincon

The function *fmincon* finds a constrained minimum of a function of several variables of the form

$$\begin{aligned} & \text{Minimize } f(x) \\ & \text{Subject to } Ax \leq b, A_{eq}x = b_{eq} \text{ (linear constraints), } lb \leq x \leq ub \end{aligned}$$

The basic syntax is

```
[x, f] = fmincon(fun, x0, A, b, Aeq, Beq, lb, ub)
```

where f is the function value at the optimum point, x_0 is the starting point, and fun is a function handle. If no bounds exist, we may use empty matrices for lb and ub or just omit them. If x_i is unbounded below, set $lb(i) = -\text{Inf}$, and set $ub(i) = \text{Inf}$ if x_i is unbounded above. Depending on the problem type or conditions, the corresponding input arguments may be omitted. For example, if there exist only inequality constraints, we can use

```
x = fmincon(fun, x0, A, b)
```

10.6.2.3 fminimax

The function *fminimax* finds a minimax solution of a function of several variables of the form

$$\begin{aligned} & \text{Minimize (Max. } f(x)) \\ & \text{Subject to } Ax \leq b, A_{eq}x = b_{eq} \text{ (linear constraints), } lb \leq x \leq ub \end{aligned}$$

The basic syntax is

```
x = fminimax(fun, x0, A, b, Aeq, Beq, lb, ub)
```

where x_0 is the starting point and fun is a function handle. If no bounds exist, we may use empty matrices for lb and ub or just omit them. For example, if there is no constraints, we can use

```
x = fminimax(fun, x0)
```

10.6.2.4 quadprog

The function *quadprog* solves quadratic programming problems of the form

$$\text{Minimize } \frac{1}{2} x^T H x + c^T x$$

$$\text{Subject to } Ax \leq b, A_{eq}x = b_{eq} \text{ (linear constraints), } lb \leq x \leq ub$$

The basic syntax is

```
[x, f] = quadprog(H, c, A, b, Aeq, beq, lb, ub, x0)
```

where f is the function value at the optimum point. If no bounds exist, we may use empty matrices for lb and ub or just omit them. For example, if there exist only inequality constraints, we can use

```
x = quadprog(H, c, A, b)
```

Example 10.26: Constrained Optimization by Built-In Functions

1. Solve the constrained linear least-squares problem given by

$$\text{Minimize } \frac{1}{2} \|Cx - d\|^2$$

$$\text{Subject to } Ax \leq b$$

where

$$A = \begin{bmatrix} -2 & 1 \\ 3 & 5 \end{bmatrix}, \quad b = \begin{bmatrix} 6 \\ 8 \end{bmatrix}, \quad C = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}, \quad d = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

2. Minimize $f(x) = \left(x_1 - \frac{1}{2}\right)^2 (x_1 + 1)^2 + 2(x_2 + 1)^2 (x_2 - 1)^2$

$$\text{Subject to } 2x_1 + 4x_2 \leq 7, -3x_1 + x_2 \leq 3$$

Set $x_0 = [0 \ 0]$.

3. Find a minimax solution of $f(x) = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 5$. Use $x_0 = 1$.

4. Minimize $f(x) = -4x_1 + x_1^2 - 2x_1x_2 + 2x_2^2$

$$\text{Subject to } 2x_1 + x_2 \leq 6, x_1 - 4x_2 \leq 0, x_1 \geq 0, x_2 \geq 0$$

Solution

1. We use the function *lsqlin*:

```
>> C = [2 0; 0 3]; d = [4; 4]; A = [-2 1; 3 5]; b = [6; 8];
>> x = lsqlin(C, d, A, b)
Optimization terminated.

x =
    1.3039
    0.8177
```

2. We can see that $A = \begin{bmatrix} 2 & 4 \\ -3 & 1 \end{bmatrix}$, $b = \begin{bmatrix} 7 \\ 3 \end{bmatrix}$. The function *fmincon* may be used:

```
>> f = @(x) (x(1)-1/2)^2*(x(1)+1)^2 + 2*(x(2)+1)^2*(x(2)-1)^2;
>> A = [2 4; -3 1]; b = [7; 3]; x0 = [0 0];
>> [x fv] = fmincon(f, x0, A, b)
x =
    0.5000    -1.0000
fv =
    2.8902e-16
```

3. We use the function *fminimax*:

```
>> f = @(x) 1/((x-0.3)^2+0.01) + 1/((x-0.9)^2+0.04) - 5; x0 = 1;
>> [x,fv] = fminimax(f, x0)
x =
    2.2928e+07
fv =
    -5.0000
```

4. The function can be expressed as

$$f(x) = \frac{1}{2} \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & -2 \\ -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -4 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

and the constraints can be written as

$$\begin{bmatrix} 2 & 1 \\ 1 & -4 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} x \leq \begin{bmatrix} 6 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We use the function *quadprog*:

```
>> H = [2 -2; -2 4]; c = [-4; 0]; A = [2 1; 1 -4; -1 0; 0 -1]; b = [6 0 0 0]';
>> [x,f] = quadprog(H, c, A, b)
x =
    2.4615
    1.0769
f =
    -6.7692
```

10.6.3 LINEAR AND MIXED-INTEGER PROGRAMMING PROBLEMS

10.6.3.1 *linprog*

The function *linprog* solves the linear programming problem of the form

Minimize $f^T x$
Subject to $Ax \leq b$, $A_{eq}x = b_{eq}$ (linear constraints), $lb \leq x \leq ub$

The basic syntax is

```
[x, fv] = linprog(f, A, b, Aeq, Beq, lb, ub, x0)
```

where *fv* is the function value at the optimum point *x*. If no bounds exist, we may use empty matrices for *lb* and *ub* or just omit them. If x_i is unbounded below, set *lb*(*i*) = -Inf, and set *ub*(*i*) = Inf if x_i is unbounded above. Depending on the problem type or conditions, the corresponding input arguments may be omitted. For example, if there exist only inequality constraints, we can use

```
[x, fv] = linprog(f, A, b)
```

10.6.3.2 intlinprog

The function *intlinprog* solves mixed-integer linear programming problems of the form

Minimize $f^T x$

Subject to $Ax \leq b$, $A_{eq}x = b_{eq}$, (linear constraints), $lb \leq x \leq ub$, x_i : integers

The basic syntax is

```
[x, fv] = intlinprog (f, intcon, A, b, Aeq, Beq, lb, ub)
```

where *intcon* is the index vector containing integer variables. If no bounds exist, we may use empty matrices for *lb* and *ub* or just omit them. If x_i is unbounded below, set *lb*(*i*) = $-\text{Inf}$, and set *ub*(*i*) = Inf if x_i is unbounded above. Depending on the problem type or conditions, the corresponding input arguments may be omitted. For example, if there exist only inequality constraints, we can use

```
[x, fv] = linprog (f, A, b)
```

Example 10.27: Linear Programming Problem by the Built-In Function

Minimize $f(x) = -50x_1 - 98x_2 - 25x_3 - 43x_4$

Subject to $6x_1 + 12x_2 + 3x_3 + 8x_4 \leq 1150$, $4x_1 + 30x_2 + 2x_3 + x_4 \leq 750$, $x_i \geq 0$ ($i = 1, \dots, 4$)

Solution

The coefficient vector is given by $f = [-50 \ -98 \ -25 \ -43]^T$ and the constraints can be expressed as

$$\begin{bmatrix} 6 & 12 & 3 & 8 \\ 4 & 30 & 2 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} x \leq \begin{bmatrix} 1150 \\ 750 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

```
>> f = [-50 -98 -25 -43]';
>> A = [6 12 3 8;4 30 2 1;-1 0 0 0;0 -1 0 0;0 0 -1 0;0 0 0 -1];
>> b = [1150 750
0 0 0 0];
>> [x, fv] = linprog (f, A, b)
Optimization terminated.

x =
128.7717
-0.0000
115.5335
3.8462
fv =
-9.4923e+03
```

Example 10.28: Mixed-Integer Programming Problem by the Built-In Function

Minimize $f(x) = -2x_1 - 3x_2$ (x_1, x_2 : integer variables)

Subject to $4x_1 + 10x_2 \leq 45$, $4x_1 + 4x_2 \leq 23$, $x_i \geq 0$ ($i = 1, 2$)

Solution

The coefficient vector is given by $f = [-2 -3]^T$ and the constraints can be expressed as

$$\begin{bmatrix} 4 & 10 \\ 4 & 4 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} x \leq \begin{bmatrix} 45 \\ 23 \\ 0 \\ 0 \end{bmatrix}$$

Since x_1 and x_2 are integer variables, $\text{intcon} = [1 2]$.

```
>> c = [-2 -3]; A = [4 10; 4 4; -1 0; 0 -1]; b = [45 23 0 0]; intcon = [1 2];
>> [x, fv] = intlinprog(c, intcon, A, b)
LP: Optimal objective value is -15.166667.
Optimal solution found.
x =
    1.0000
    4.0000
fv =
    -14
```

PROBLEMS

- 10.1** A metallic ball released from a height h at an angle θ and velocity v with respect to the horizontal in a gravitational field g travels a distance d when it hits the ground. The distance d is given by

$$d = \left(\frac{v \sin \theta}{g} + \sqrt{\frac{2h}{g} + \left(\frac{v \sin \theta}{g} \right)^2} \right) v \cos \theta$$

Use the Fibonacci search method to find θ for which the distance d is a maximum and to determine the maximum distance d .

Data: $h = 60$ m, $v = 80$ m, $g = 9.8$ m/sec 2 , $n = 30$, $a = 0$, $b = 1.6$.

- 10.2** A company plans to borrow $S \times 1000$ dollars to build a new plant on equal yearly installments over the next n years. The interest charged is $r_e = e_1 + e_2 S$. The money earned can be reinvested at a rate of m_r per year. The expected future value of the return is given by

$$f_R = P \left(1 - e^{-wS} \right)$$

And the future value of the payment is given by

$$f_P = \left[\frac{(1 + m_r)^n - 1}{m_r} \right] \left[\frac{r_e (1 + r_e)^n}{(1 + r_e)^n - 1} \right] S$$

Use the golden section search method to determine the value of S to be borrowed for the maximum future value of profit F , which is given by $F = f_R - f_P$.

Data: $e_1 = 0.046$, $e_2 = 0.00028$, $m_r = 0.061$, $P = 500 \times 10^3$ \$, $w = 0.012/1000$, $n = 6$.

- 10.3** Use Brent's search method to find the minimum of $f(x) = 87.5(1 - x^3)^2 + (1 - x^2) + 3(1 - x)^2$. The initial point can be specified as $x_1 = 0.01$. Let the step size be $h = 0.2$ and the stopping criterion be 1×10^{-6} .

- 10.4** Use the Shubert–Piyavskii method to find the maximum of $f(x) = \sum_{j=1}^5 j \sin((j+1)x + j)$. Use the following conditions: initial interval = $[-10, 10]$, Lipschitz constant $C = 80$, stopping criterion = 1×10^{-6} , maximum number of function evaluations = 2000.³⁰
- 10.5** Use the steepest descent method to find the minimum of the function given by

$$f(x) = x_1^2 - x_1 x_2 - 4x_1 + x_2^2 - x_2$$

The initial point can be specified as $[0, 0]$. Use the initial step size of $\alpha_0 = 2$. The stopping criterion and the maximum number of function evaluations can be set as 1×10^{-6} and 10,000, respectively.

- 10.6** Use the steepest descent method to find the minimum of Wood's function given by

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1 \left[(x_2 - 1)^2 + (x_4 - 1)^2 \right] + 19.8(x_2 - 1)(x_4 - 1)$$

The initial point can be specified as $[-3, -1, -3, -1]$. Use the initial step size of $\alpha_0 = 2$. The stopping criterion and the maximum number of function evaluations can be set as 1×10^{-6} and 10,000, respectively.

- 10.7** Use Newton's method to find the minimum of the function given by

$$f(x) = \begin{bmatrix} 2x_1 - x_2 - 4 \\ -x_1 + 2x_2 - 1 \end{bmatrix}$$

The initial point can be specified as $[0, 0]$. The stopping criterion and the maximum number of function evaluations can be set as 1×10^{-6} and 1000, respectively.

- 10.8** Wood's function is given by

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1 \left[(x_2 - 1)^2 + (x_4 - 1)^2 \right] + 19.8(x_2 - 1)(x_4 - 1)$$

Use the conjugate gradient method to find the minimum of Wood's function. The initial point can be specified as $[-3, -1, -3, -1]$. Use the initial step size of $\alpha_0 = 2$. The stopping criterion and the maximum number of function evaluations can be set as 1×10^{-6} and 10,000, respectively.

- 10.9** Use the Davidon–Fletcher–Powell (DFP) method to find the minimum of Rosenbrock's function given by

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

The initial point can be specified as $[-1.2, 1]$. Use the initial step size of $\alpha_0 = 1$. The stopping criterion and the maximum number of function evaluations can be set as 1×10^{-6} and 1000, respectively.

- 10.10** Himmelblau's function is given by

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

1. Construct mesh and contour plots for this function in the range of $-6 \leq x_1, x_2 \leq 6$.
 2. Determine the minimum using the Davidon–Fletcher–Powell (DFP) method. Use the initial step size of $\alpha_0 = 1$ and set the initial point as $[0, 0]$. The stopping criterion and the maximum number of function evaluations can be set as 1×10^{-6} and 1000, respectively.
- 10.11** Find the solution of the following LP using the two-phase method:
- Minimize $f(x) = 2x_1 + 6x_2$
 Subject to $-x_1 + x_2 \leq 1$, $2x_1 + x_2 \leq 2$, $x_1 \geq 0$, $x_2 \geq 0$
- 10.12** A farmer has the choice of producing tomatoes, green peppers, or cucumbers on his 200 acre farm. A total of 480 man-days of labor is available. Assuming fertilizer cost is the same for all products, determine the optimum crop combination using the two-phase simplex method.

	Yield (\$/Acre)	Labor (Man-Days/Acre)
Tomatoes	410	6
Green peppers	350	8
Cucumbers	390	7

- 10.13** Find the solution of the following LP using the interior point method:
- Minimize $f(x) = -x_1 - 2x_2 - x_3$
 Subject to $2x_1 + x_2 - x_3 \leq 2$, $2x_1 - x_2 + 5x_3 \leq 6$, $4x_1 + x_2 + x_3 \leq 9$, $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$
- 10.14** Find the solution of the following constrained minimization problem using Rosen's gradient projection method:
- Minimize $f(x) = -x_1 x_2 x_3$
 Subject to $0 \leq x_1 + 2x_2 + x_3 \leq 70$, $0 \leq x_1, x_2, x_3 \leq 40$
 As a starting point, use $x^0 = [5, 5, 5]$. Note that this point satisfies all constraints.
- 10.15** Find the solution of the following constrained minimization problem using the generalized reduced gradient method:
- Minimize $f(x) = -(1.2x_1 + 3x_2)$
 Subject to $h(x) = x_1^2 + 6x_2^2 - 1 \leq 0$, $0 \leq x_1, x_2 \leq 10$
 The inequality constraint can be converted to equality by introducing slack variable x_3 as follows:

$$g(x) = x_1^2 + 6x_2^2 + x_3 - 1 = 0$$

- As a starting point, use $x^0 = [0, 0, 1]$. Note that this point satisfies all constraints. Set $kmax = 1000$ and $crit = 1 \times 10^{-4}$.
- 10.16** Find the solution of the following constrained minimization problem using the sequential quadratic programming method³¹:
- Minimize $f(x) = x_1^2 + x_2$
 Subject to $h(x) = x_1^2 + x_2^2 = 9$, $1 \leq x_1 \leq 5$, $2 \leq x_2 \leq 4$
 As a starting point, use $x^0 = [3, 3]$. The initial values of Lagrangian multipliers can be set as 1 and 4. Set $crit = 1 \times 10^{-6}$.

- 10.17** Find the solution of the following constrained minimization problem using the sequential quadratic programming method:

$$\text{Minimize } f(x) = x_1^2 + x_2^2 + 2.3x_3^2 - 1.2x_4^2 - 4x_1 - 6x_2 - 20x_3 + 6x_4 + 100$$

Subject to

$$h_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_1 - x_2 + x_3 - x_4 \leq 7$$

$$h_2(x) = x_1^2 + 2x_2^2 + x_3^2 + 2x_4^2 - x_1 - x_4 \leq 11$$

$$h_3(x) = 2x_1^2 + x_2^2 + x_3^2 + 2x_1 - x_2 - x_4 \leq 6, \quad -100 \leq x_1, x_2, x_3, x_4 \leq 100$$

As a starting point, we can use $x^0 = [0 \ 0 \ 0 \ 0]^T$. The initial values of Lagrangian multipliers can be set as 1 and 4. Set crit = 1×10^{-6} .

- 10.18** Find the minimum of each of the given functions using the cyclic coordinate search method.

$$1. \quad f(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \quad (\text{Rosenbrock's function}), \quad x^0 = [-1, 1], \quad \text{crit} = 1 \times 10^{-4}.$$

$$2. \quad f(x) = x_1^2 + 2x_2^2 + 2x_1x_2, \quad x^0 = [0.5, 1], \quad \text{crit} = 1 \times 10^{-4}.$$

- 10.19** Find the minimum of $f(x) = x_1^2 + 2x_2^2 + 2x_1x_2$ using the Nelder and Mead's simplex method. Use $x^0 = [0.5, 1]$ and crit = 1×10^{-6} .

- 10.20** Find the minimum of Rosenbrock's function using the simulated annealing method:

$$f(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \quad (-5 \leq x_1, x_2 \leq 5)$$

Set the initial temperature as $T = 100$, the temperature reduction factor $rp = 0.8$, the step reduction parameter $rs = 0.9$, and crit = 1×10^{-8} .

- 10.21** Find the minimum of the given function $f(x)$ using the simulated annealing method³²:

$$f(x) = x_1^4 - 16x_1^2 - 5x_1 + x_2^4 - 16x_2^2 - 5x_2 \quad (-5 \leq x_1, x_2 \leq 5)$$

Set the initial temperature as $T = 100$, the temperature reduction factor $rp = 0.8$, the step reduction parameter $rs = 0.9$, and crit = 1×10^{-8} .

- 10.22** Find the minimum of Powell's function using the genetic algorithm:

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \quad -5 \leq x_1, x_2, x_3, x_4 \leq 5$$

Set nb = 8, ps = 50, ng = 50, and mp = 0.04.

- 10.23** Find the solution of the following minimization problem using the branch and bound algorithm³³:

$$\text{Minimize } f(x) = 65x_1 + 57x_2 + 42x_3 + 20x_4$$

Subject to $4x_1 + 5x_2 + 6x_3 + 10x_4 \leq 12$, $x_1 + x_2 = 1$, $x_3 + x_4 = 1$ ($x_i = 0$ or 1)

- 10.24** Find the solution of the following minimization problem using the zero-one programming method:

$$\text{Minimize } f(x) = 4x_1 + 5x_2 + 3x_3$$

Subject to $-x_1 + 3x_2 + 6x_3 \leq 6$, $-2x_1 - 3x_2 - 3x_3 \leq -2$, $-x_1 - x_3 \leq -1$, $0 \leq x_1, x_2$ ($x_i = 0$ or 1)

- 10.25** Suppose that the annual cost $C(d)$ (\$) of a pipeline can be expressed as a function of the pipe diameter d (in.) as follows:

$$C(d) = 500 + 240d^{1.5} + \frac{2}{d^{2.5}} + \frac{27}{d^4}$$

1. Plot $C(d)$ versus d .
2. Find the optimal value of d using a MATLAB's built-in function.
3. If d should be within the interval $[1, 3]$, what will be the optimum annual cost? Use a MATLAB's built-in function that solves constrained optimization problem.

REFERENCES

1. Brent, R. P., *Algorithms of Minimization without Derivatives*, Prentice-Hall, Englewood Cliffs, NJ, pp. 47–60, 1973.
2. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, pp. 75–76, 2011.
3. Fletcher, R. and M. J. D. Powell, A rapidly convergent descent method for minimization, *The Computer Journal*, 6, 163–168, 1963.
4. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, p. 108, 2011.
5. Edgar, T. F., D. M. Himmelblau, and L. S. Lasdon, *Optimization of Chemical Processes*, 2nd ed., McGraw-Hill, New York, NY, p. 197, 2001.
6. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, p. 114, 2011.
7. Rao, S. S., *Engineering Optimization: Theory and Practice*, 4th ed., John Wiley & Sons, Inc., Hoboken, NJ, pp. 150–152, 2009.
8. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, pp. 132–135, 2011.
9. Karmarkar, N., A new polynomial time algorithm for linear programming, *Combinatorica*, 4, 373–395, 1984.
10. Lindfield, G. G. and J. E. T. Penny, *Numerical Methods*, 3rd ed., Academic Press, Waltham, MA, pp. 374–375, 2012.
11. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, p. 220, 2011.
12. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, p. 226, 2011.
13. Rao, S. S., *Engineering Optimization*, 4th ed., John Wiley & Sons, Inc., Hoboken, NJ, pp. 413–414, 2009.
14. Lasdon, L. et al., Design and testing of a generalized relaxed gradient code for nonlinear programming, *ACM Transactions on Mathematical Software*, 4(1), 34–50, 1978.
15. Antoniou, A. and W.-S. Lu, *Practical Optimization*, Springer, New York, NY, pp. 514–515, 2007.
16. Antoniou, A. and W.-S. Lu, *Practical Optimization*, Springer, New York, NY, p. 516, 2007.
17. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, pp. 295–296, 2011.
18. Hooke, R. and T. A. Jeeves, Direct search solution of numerical and statistical problems, *Journal of the ACM*, 8, 212–229, 1961.
19. Rosenbrock, H. H., An automatic method for finding the greatest or least value of a function, *The Computer Journal*, 3, 175–184, 1960.
20. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, pp. 309–311, 2011.
21. Nelder, J. A. and R. Mead, A simplex method for function minimization, *The Computer Journal*, 7, 308–313, 1965.
22. Corana, A., M. Marchesi, C. Martini, and S. Ridella, Minimizing multimodal functions of continuous variables with simulated annealing algorithm, *ACM Transactions on Mathematical Software*, 13, 262–280, 1987.

23. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, pp. 317–318, 2011.
24. Venkataraman, P., *Applied Optimization with MATLAB Programming*, 2nd ed., John Wiley & Sons, Inc., Hoboken, NJ, pp. 458–459, 2009.
25. Yang, W. Y., W. Cao, T.-S. Chung, and J. Morris, *Applied Numerical Methods Using MATLAB*, Wiley Interscience, Hoboken, NJ, p. 338, 2005.
26. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, p. 322, 2011.
27. Rao, S. S., *Engineering Optimization: Theory and Practice*, 4th ed., John Wiley & Sons, Inc., Hoboken, NJ, p. 610, 2009.
28. Venkataraman, P., *Applied Optimization with MATLAB Programming*, 2nd ed., John Wiley & Sons, Inc., Hoboken, NJ, pp. 416–417, 2009.
29. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, p. 368, 2011.
30. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, p. 77, 2011.
31. Antoniou, A. and W.-S. Lu, *Practical Optimization*, Springer, New York, NY, p. 517, 2007.
32. Yang, W. Y., W. Cao, T.-S. Chung, and J. Morris, *Applied Numerical Methods Using MATLAB*, Wiley Interscience, Hoboken, NJ, pp. 337–338, 2005.
33. Belegundu, A. D. and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed., Cambridge University Press, New York, NY, pp. 366–367, 2011.

Appendix A: Supplementary Programs

A.1 PROGRAMS TO COMPUTE PHYSICAL PROPERTIES OF WATER AND STEAM

The MATLAB® programs presented below are the supplementary programs to calculate physical properties of water and steam using the IAPWS-IF97 (International Association for Properties of Water and Steam Formulation for Industrial Use) formula*:

propPT.m

```
function [P,T,v,u,s,h] = propPT(P,T,rg)
% input
%   P: pressure (MPa)
%   T: temperature (K)
%   rg: region(1, 2, 4)
% output
%   P: pressure (MPa)
%   T: temperature (K)
%   v: volume
%   u: internal energy
%   s: entropy
%   h: enthalpy

R = 0.461526;
switch rg
    case {1,4}
        Ps = P/16.53; tau = 1386/T; % P*=16.53MPa, T*=1386K
        v = (Ps*rg1dgdp(Ps,tau)*R.* T)/(P*10^3);
        u = (tau*rg1dgdp(Ps,tau)-Ps* rg1dgdp(Ps,tau))*R*T;
        s = (tau*rg1dgdp(Ps,tau) - rg1eqn(Ps,tau))*R;
        h = (tau*rg1dgdp(Ps,tau))*R*T;
    case 2
        Ps = P/(1); tau = 540/T;
        [g0,gPs0,gr,gPs1,gt0,gtr] = rg2eqn(Ps,tau);
        v = Ps*(gPs0+gPs1)*R*T/(P*10^3);
        u = R*T*(tau*(gt0+gtr) - Ps*(gPs0+gPs1));
        s = R*(tau*(gt0+gtr) - (g0+gr)); h = tau*(gt0+gtr)*R*T;
    otherwise
        P = 0;T = 0;v = 0;u = 0;s = 0;h = 0;
    end
end
```

rg1dgdp.m

```
function dgdp = rg1dgdp(Ps,tau)
% differentiate the basic equation for region 1 (g/RT) with respect to P/P*
% input
%   Ps: P/P* (P*=16.53MPa), tau: T*/T (T*=1386K)
% output
%   dgdp: results of differentiation
```

* Revised Release of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, August 2007, IAPWS, Lucerne, Switzerland (<http://www.iapws.org>).

```
% Parameters of the basic equation of region 1 (Ii, Ji and ni)
% i ranges from 1 to 34
[I,J,n] = rg1IJn;
% perform differentiation
ind = 1:34; dgdpT = sum(-n(ind).*I(ind).* (7.1-Ps).^(I(ind)-1).* 
(tau-1.222).^J(ind));
end
```

rg1dgdT.m

```
function dgdT = rg1dgdT(Ps,tau)
% differentiate the basic equation for region 1 (g/RT) with respect to T*/T
% input
% Ps: P/P* (P*=16.53MPa), tau: T*/T (T*=1386K)
% output
% dgdT: results of differentiation

% Parameters of the basic equation of region 1 (Ii, Ji and ni)
% i ranges from 1 to 34
[I,J,n] = rg1IJn;
% perform differentiation
ind = 1:34; dgdT = sum(n(ind).* (7.1-Ps).^I(ind).*J(ind).* 
(tau-1.222).^(J(ind)-1));
end
```

rg1eqn.m

```
function gfe = rg1eqn(Ps,tau)
% Calculation of the basic equation for region 1
% input
% Ps: P/P* (P*=16.53MPa)
% tau: T*/T (T*=1386K)
% output
% gfe: Gibbs free energy (dimensionless)

% Parameters of the basic equation of region 1 (Ii, Ji and ni)
% i ranges from 1 to 34
[I,J,n] = rg1IJn;
% Calculation of dimensionless Gibbs free energy
ind = 1:34; gfe = sum(n(ind).* (7.1 - Ps).^I(ind).* (tau - 1.222).^J(ind));
end
```

rg2eqn.m

```
function [g0,gPs0,gr,gPsR,gt0,gtr] = rg2eqn(Ps,tau)
% Calculation of the basic equation for region 2
% input
% Ps: P/P* (P*=1MPa), tau: T*/T (T*=540K)
% output
% g0: ln(pi) + sum(n_i^o*tau^Ji0)
% gPs0: 1/pi + 0
% gr: gamma^r
% gPsR: partial differentiation of gamma^r with respect to pi
% gt0: gamma_tau^o
% gtr: partial differentiation of gamma^r with respect to tau
% Parameters of the basic equation of region 2 (Ii, Ji and ni)
% J0,n0: ideal gas part (o)
% I,J,n: residual part (r) (i ranges from 1 to 43)
```

```
[J0,n0,I,J,n] = rg2IJn; ind1 = 1:9; ind2 = 1:43;
g0 = log(Ps) + sum(n0(ind1).*tau).^J0(ind1)); % gamma_0
gPs0 = 1/Ps + 0;
gr = sum(n(ind2).*Ps.^I(ind2).*tau.^0.5).^J(ind2)); % gamma^r
gPsr = sum(n(ind2).*I(ind2).*Ps.^I(ind2-1).*tau.^0.5).^J(ind2));
gt0 = 0 + sum(n0(ind1).*J0(ind1).*tau.^J0(ind1-1)); % gamma_tau^0
gtr = sum(n(ind2).*Ps.^I(ind2).*J(ind2).*tau.^0.5.^J(ind2-1));
end
```

rg1IJn.m

```
function [I,J,n] = rg1IJn
% Parameters of the basic equation of region 1 (Ii, Ji and ni)
% i ranges from 1 to 34
```

```
I(1) = 0; J(1) = -2; n(1) = 0.14632971213167; % i = 1
I(2) = 0; J(2) = -1; n(2) = -0.84548187169114; % i = 2
I(3) = 0; J(3) = 0; n(3) = -0.37563603672040*10^1; % i = 3
I(4) = 0; J(4) = 1; n(4) = 0.33855169168385*10^1; % i = 4
I(5) = 0; J(5) = 2; n(5) = -0.95791963387872; % i = 5
I(6) = 0; J(6) = 3; n(6) = 0.15772038513228; % i = 6
I(7) = 0; J(7) = 4; n(7) = -0.16616417199501*10^-1; % i = 7
I(8) = 0; J(8) = 5; n(8) = 0.81214629983568*10^-3; % i = 8
I(9) = 1; J(9) = -9; n(9) = 0.28319080123801*10^-3; % i = 9
I(10) = 1; J(10) = -7; n(10) = -0.60706301565874*10^-3; % i = 10
I(11) = 1; J(11) = -1; n(11) = -0.18990068218419*10^-1; % i = 11
I(12) = 1; J(12) = 0; n(12) = -0.32529748770505*10^-1; % i = 12
I(13) = 1; J(13) = 1; n(13) = -0.21841717175414*10^-1; % i = 13
I(14) = 1; J(14) = 3; n(14) = -0.52838357969930*10^-4; % i = 14
I(15) = 2; J(15) = -3; n(15) = -0.47184321073267*10^-3; % i = 15
I(16) = 2; J(16) = 0; n(16) = -0.30001780793026*10^-3; % i = 16
I(17) = 2; J(17) = 1; n(17) = 0.47661393906987*10^-4; % i = 17
I(18) = 2; J(18) = 3; n(18) = -0.44141845330846*10^-5; % i = 18
I(19) = 2; J(19) = 17; n(19) = -0.72694996297594*10^-15; % i = 19
I(20) = 3; J(20) = -4; n(20) = -0.31679644845054*10^-4; % i = 20
I(21) = 3; J(21) = 0; n(21) = -0.28270797985312*10^-5; % i = 21
I(22) = 3; J(22) = 6; n(22) = -0.85205128120103*10^-9; % i = 22
I(23) = 4; J(23) = -5; n(23) = -0.22425281908000*10^-5; % i = 23
I(24) = 4; J(24) = -2; n(24) = -0.65171222895601*10^-6; % i = 24
I(25) = 4; J(25) = 10; n(25) = -0.14341729937924*10^-12; % i = 25
I(26) = 5; J(26) = -8; n(26) = -0.40516996860117*10^-6; % i = 26
I(27) = 8; J(27) = -11; n(27) = -0.12734301741641*10^-8; % i = 27
I(28) = 8; J(28) = -6; n(28) = -0.17424871230634*10^-9; % i = 28
I(29) = 21; J(29) = -29; n(29) = -0.68762131295531*10^-18; % i = 29
I(30) = 23; J(30) = -31; n(30) = 0.14478307828521*10^-19; % i = 30
I(31) = 29; J(31) = -38; n(31) = 0.26335781662795*10^-22; % i = 31
I(32) = 30; J(32) = -39; n(32) = -0.11947622640071*10^-22; % i = 32
I(33) = 31; J(33) = -40; n(33) = 0.18228094581404*10^-23; % i = 33
I(34) = 32; J(34) = -41; n(34) = -0.93537087292458*10^-25; % i = 34
end
```

rg2IJn.m

```
function [J0,n0,I,J,n] = rg2IJn
% Parameters of the basic equation of region 2 (Ii, Ji and ni)
% J0,n0: ideal gas part (o)
% I,J,n: residual part (r) (i ranges from 1 to 43)
```

```

% ideal gas part (o)
J0(1) = 0; n0(1) = -0.96927686500217*10^1; % i = 1
J0(2) = 1; n0(2) = 0.10086655968018*10^2; % i = 2
J0(3) = -5; n0(3) = -0.56087911283020*10^-2; % i = 3
J0(4) = -4; n0(4) = 0.71452738081455*10^-1; % i = 4
J0(5) = -3; n0(5) = -0.40710498223928; % i = 5
J0(6) = -2; n0(6) = 0.14240819171444*10^1; % i = 6
J0(7) = -1; n0(7) = -0.43839511319450*10^1; % i = 7
J0(8) = 2; n0(8) = -0.28408632460772; % i = 8
J0(9) = 3; n0(9) = 0.21268463753307*10^-1; % i = 9

% residual part (r)
I(1) = 1; J(1) = 0; n(1) = -0.17731742473213*10^-2; % i = 1
I(2) = 1; J(2) = 1; n(2) = -0.17834862292358*10^-1; % i = 2
I(3) = 1; J(3) = 2; n(3) = -0.45996013696365*10^-1; % i = 3
I(4) = 1; J(4) = 3; n(4) = -0.57581259083432*10^-1; % i = 4
I(5) = 1; J(5) = 6; n(5) = -0.50325278727930*10^-1; % i = 5
I(6) = 2; J(6) = 1; n(6) = -0.33032641670203*10^-4; % i = 6
I(7) = 2; J(7) = 2; n(7) = -0.18948987516315*10^-3; % i = 7
I(8) = 2; J(8) = 4; n(8) = -0.39392777243355*10^-2; % i = 8
I(9) = 2; J(9) = 7; n(9) = -0.43797295650573*10^-1; % i = 9
I(10) = 2; J(10) = 36; n(10) = -0.26674547914087*10^-4; % i = 10
I(11) = 3; J(11) = 0; n(11) = 0.20481737692309*10^-7; % i = 11
I(12) = 3; J(12) = 1; n(12) = 0.43870667284435*10^-6; % i = 12
I(13) = 3; J(13) = 3; n(13) = -0.32277677238570*10^-4; % i = 13
I(14) = 3; J(14) = 6; n(14) = -0.15033924542148*10^-2; % i = 14
I(15) = 3; J(15) = 35; n(15) = -0.40668253562649*10^-1; % i = 15
I(16) = 4; J(16) = 1; n(16) = -0.78847309559367*10^-9; % i = 16
I(17) = 4; J(17) = 2; n(17) = 0.12790717852285*10^-7; % i = 17
I(18) = 4; J(18) = 3; n(18) = 0.48225372718507*10^-6; % i = 18
I(19) = 5; J(19) = 7; n(19) = 0.22922076337661*10^-5; % i = 19
I(20) = 6; J(20) = 3; n(20) = -0.16714766451061*10^-10; % i = 20
I(21) = 6; J(21) = 16; n(21) = -0.21171472321355*10^-2; % i = 21
I(22) = 6; J(22) = 35; n(22) = -0.23895741934104*10^2; % i = 22
I(23) = 7; J(23) = 0; n(23) = -0.59059564324270*10^-17; % i = 23
I(24) = 7; J(24) = 11; n(24) = -0.12621808899101*10^-5; % i = 24
I(25) = 7; J(25) = 25; n(25) = -0.38946842435739*10^-1; % i = 25
I(26) = 8; J(26) = 8; n(26) = 0.11256211360459*10^-10; % i = 26
I(27) = 8; J(27) = 36; n(27) = -0.82311340897998*10^1; % i = 27
I(28) = 9; J(28) = 13; n(28) = 0.19809712802088*10^-7; % i = 28
I(29) = 10; J(29) = 4; n(29) = 0.10406965210174*10^-18; % i = 29
I(30) = 10; J(30) = 10; n(30) = -0.10234747095929*10^-12; % i = 30
I(31) = 10; J(31) = 14; n(31) = -0.10018179379511*10^-8; % i = 31
I(32) = 16; J(32) = 29; n(32) = -0.80882908646985*10^-10; % i = 32
I(33) = 16; J(33) = 50; n(33) = 0.10693031879409; % i = 33
I(34) = 18; J(34) = 57; n(34) = -0.33662250574171; % i = 34
I(35) = 20; J(35) = 20; n(35) = 0.89185845355421*10^-24; % i = 35
I(36) = 20; J(36) = 35; n(36) = 0.30629316876232*10^-12; % i = 36
I(37) = 20; J(37) = 48; n(37) = -0.42002467698208*10^-5; % i = 37
I(38) = 21; J(38) = 21; n(38) = -0.59056029685639*10^-25; % i = 38
I(39) = 22; J(39) = 53; n(39) = 0.37826947613457*10^-5; % i = 39
I(40) = 23; J(40) = 39; n(40) = -0.12768608934681*10^-14; % i = 40
I(41) = 24; J(41) = 26; n(41) = 0.73087610595061*10^-28; % i = 41
I(42) = 24; J(42) = 40; n(42) = 0.55414715350778*10^-16; % i = 42
I(43) = 24; J(43) = 58; n(43) = -0.94369707241210*10^-6; % i = 43
end

```

A.2 PROGRAM FOR MULTIPLE-EFFECT EVAPORATION (ENGLISH UNITS)

```

function res = multievap(evdat)
% Calculation of multiple-effect evaporator

% Data
crit = 1e-3;
Tc = 647.096; % critical temperature of H2O (K)
Pc = 22064000; % critical pressure of H2O (Pa)
T0 = 273.15;
xf = evdat.xf; xpn = evdat.xp; mf = evdat.mf;
Tf = (evdat.Tf-32)/1.8 + 273.15; % F->K
Ps = evdat.Ps*6894.757; % psia->Pa (steam pressure)
Pn = evdat.Pn*6894.757; % psia->Pa (pressure of final evaporator)
U = evdat.U; % vector of overall heat transfer coefficients
cf = 1/2326; % J/kg->Btu/lb
n = length(U);

% Saturation temperature
a = [-7.85951783 1.84408259 -11.7866497 22.6807411 -15.9618719
      1.80122502];
gs = @(x) a(1)*x + a(2)*x^1.5 + a(3)*x^3 + a(4)*x^3.5 + a(5)*x^4 +...
      a(6)*x^7.5 - (1-x)*log(Ps/Pc);
gn = @(x) a(1)*x + a(2)*x^1.5 + a(3)*x^3 + a(4)*x^3.5 + a(5)*x^4 +...
      a(6)*x^7.5 - (1-x)*log(Pn/Pc);
xs = fzero(gs,0.5); xn = fzero(gn,0.5); Ts = Tc*(1 - xs); % steam
temperature
TsF = (Ts - 273.15)*1.8+32; % K->F
Tn = Tc*(1 - xn); % temperature of final evaporator
TnF = (Tn - 273.15)*1.8+32; % K->F

% Enthalpy
sts = satsteam(Ts-T0); % saturated steam
stv = satsteam(Tn-T0); % final evaporator
stf = satsteam(Tf-T0); % feed
Hv0 = sts.hV*cf; hp0 = sts.hL*cf;
Hvn = stv.hV*cf; hpn = stv.hL*cf; hf = stf.hL*cf;

% Material balance and temperature difference
mpn = xf*mf/xpn; % Btu/hr
dTtotal = TsF - TnF; % K->F
sumU = sum(1./U); dT = (1./U)*dTtotal/sumU; % F
critA = 10; oldA = 10*ones(1,n); iter = 0;
while critA >= crit
    T(1) = TsF - dT(1);
    for j = 2:n, T(j) = T(j-1) - dT(j); end
    TK = (T-32)/1.8 + 273.15; % F->K
    for j = 1:n
        sprop(j) = satsteam(TK(j)-T0);
        Hv(j) = sprop(j).hV * cf; hp(j) = sprop(j).hL * cf;
    end
    % Balance equations
    evM = [Hv0-hp0 hp(1)-Hv(1) 0;...
            0 Hv(1)+hp(2)-2*hp(1) hp(2)-Hv(2) ;...
            0 Hv(3)-hp(2) Hv(2)+Hv(3)-2*hp(2) ];

```

```
evb = [hp(1)-hf; hp(2)-hp(1); Hv(3)-hp(2)+(hp(3)-Hv(3))*xf/xpn]*mf;
mv = evM\evb; q(1) = mv(1)*(Hv0-hp0);
for j = 2:n, q(j) = mv(j)*(Hv(j)-hp(j)); end
Area = q./ (U.*dT); avgA = sum(Area)/n;
critA = sum(abs(Area - oldA)); dT = dT.*Area/avgA;
if abs(sum(dT) - dTtotal) >= crit, dT = dT*dTtotal/sum(dT); end
iter = iter + 1; oldA = Area;
end

% Results
res.T = T; % temperature vector
res.A = Area; % vector of evaporator areas
res.mv = mv; % vector of vapor flow rates
res.iter = iter; % number of iterations
end
```

Appendix B: List of Programs

CHAPTER 2

bcprob	Definition of differential equations
bcval	Specification of boundary conditions
biom	Definition of the system of stiff differential equations
catfb	Modeling equations for a fluidized bed reactor
cstm	Definition of stirred-tank system
ethanrxn	Solution of the steam decomposition reaction of ethane
expfn	Definition of differential equations
pdeTde	One-dimensional parabolic PDE
pdeTic	Specification of initial conditions
pdeTbc	Specification of boundary conditions
rxnfun	Definition of the system of nonlinear equations for reaction equilibrium
vdpeqn	Definition of the van der Pol equation
zode	2nd-order differential equation
zobc	Specification of boundary conditions

CHAPTER 3

compID	Assignment of characteristic numbers for each compound
condG	Heat conductivity of low-pressure gas
condL	Heat conductivity of saturated liquid
corstsg	Surface tension using corresponding state relationship
costald	COSTALD density estimation method
denL	Density of saturated liquid
H2OPT	Properties of water
hcapG	Heat capacity of gas
hcapL	Heat capacity of liquid
hcRB	Rowlinson/Bondi estimation method for liquid heat capacity
hform	Heat of formation of ideal gas at low temperature
humidest	Absolute humidity and vapor pressure
hvapn	Calculation of heat of vaporization
gdiffc	Liquid-phase diffusion coefficient
gfree	Gibbs free energy of formation
ngasZ	Compressibility factor for natural hydrocarbon gas
pitzersg	Surface tension by Pitzer's corresponding state relationship
propPT	Properties of water at given P and T
prVp	Vapor pressure of saturated liquid
satsteam	Properties of saturated steam
surtL	Surface tension of liquid
vhwagner	Estimation of vaporization enthalpy using the Wagner equation
visG	Vapor viscosity
visL	Viscosity of saturated liquid

vpRM	Estimation of vapor pressure using the Rarey/Moller correlation
vpwagner	Estimation of vapor pressure using the Wagner correlation

CHAPTER 4

binVLE	Binary VLE using the modified Raoult's law
bubbp	Bubble P calculation
bubbleT	Calculation of the bubble point for a mixture
bzacfun	Vapor pressure differential equation
bzacP	Vapor composition and activity coefficients for benzene/acetic acid system
compBCD	Constants for the Lee–Kesler equation
compVPeqn	Comparison of vapor pressure equations
cubicEOSZ	Compressibility factor and molar volume by cubic equations of state
delH	Enthalpy change and average heat capacity
delHS	Change of enthalpy and entropy due to state change
deptfun	Departure function
dewpf	VLE nonlinear equations
dewpf2	Nonlinear equations
dewpt	Dew T calculation
falp0	Nonlinear equations for bubble-point calculation
falp1	Nonlinear equations for dew-point calculation
flashBubP	Flash calculation
flashdrum	Calculation of flash drum
ibweqf	VLLE nonlinear equations
ibweqf2	Nonlinear equations
khmix	Enthalpy of mixture
phigas	Fugacity coefficients
phimix	Fugacity coefficients of mixture using cubic equations of state
resbutane	Residue properties
unifgam	Activity coefficients by the UNIFAC method
usekhmix	Use of the khmix function
useunifgam	Calculation of activity coefficient using the unifgam function
virialeOS	Virial equation of state
wilact	Parameters for the Wilson equation
wilsonpar	Calculation of Wilson equation parameters for binary system
zLK	Compressibility factor by the Lee–Kesler equation

CHAPTER 5

cltank	Differential equations for a closed tank
compflow	Flow of compressible fluid
dpcatbed	Pressure drop in a fluidized bed
dwfun	Pipe diameter
eqplen	Equivalent length and actual length of a pipe
frfactor	Comparison of friction factor correlation equations
hzannpipe	Velocity profile for Newtonian flow through a horizontal annular pipe

hzpipe	Average velocity and velocity profile for horizontal laminar flow
nnDfun	Diameter of a horizontal pipe for non-Newtonian fluid flow
nnhzpipe	Average velocity and velocity profile for the horizontal laminar flow of a non-Newtonian fluid
pipnet	System of nonlinear equations for a pipe network
pnetq	Nonlinear equations for a pipeline network
powfun	Water transportation using a pump
qrmT	Properties as a function of temperature
twophdP	Pressure drop in two-phase flow
twophmod	Calculation of Y_G in each flow region
twophreg	Baker parameter for two-phase flow region
usectank	Use of the function cltank
usepipnet	Use of the function pipnet
vtfun	Nonlinear equation for terminal velocity
vtwall	Average velocity and velocity profile for the vertical surface flow of a Newtonian fluid
vwfig	Velocity and flow rate according to the pipe inside diameter and pipe length
vwfun	Velocity function

CHAPTER 6

adbpf	Differential equation for conversion rate
adfun	System of differential equations for an acetone decomposition reaction
cstrmult	System of nonlinear multiple reactions
exbco	Differential equation for a butane conversion reaction
exocstr	Exothermic reaction in a CSTR
exocstr2	Steady state of exothermic reactions in a CSTR
geferm	System of differential equations for a biochemical reaction
mbrmult	System of equations for multiple reactions in a membrane reactor
mcatb	Decomposition reaction of gas oil
membrx	System of differential equations for a membrane reactor
microrx	Differential equation for a microporous reactor
pbconv	Differential equation for a packed-bed catalytic reactor
pbrmf	System of differential equations for a packed-bed catalytic reactor
pbrmult	System of nonlinear equations for multiple reactions in a packed-bed catalytic reactor
pbrpar	Parameter estimation
rateconst	Activation energy and reaction rate frequency factor
regorder	System of nonlinear equations
rnk	Parameters for a batch reaction
semibrx	Semibatch reaction equation
ser3v	Solution of system of differential equations
serrb	Solution of a differential reaction equation
usead	Temperature and conversion rate as a function of pressure and change in the feed rate
useadbpf	Use of the function adbpf
usecstrmult	Solution of the function cstrmult
useexbco	Solution of the function exbco

usegeferm	Solution of the function geferm
usembrmult	Solution of the function mbrmult
usemembrx	Solution of the function membrx
usemicrorx	Solution of the function microrx
usepbconv	Solution of the function pbconv
usepbrmult	Solution of the function pbrmult
usesemibrx	Solution of the function semibrx

CHAPTER 7

absbz	Change of benzene composition with time in an absorption column
absf	Downward laminar flow function
bindistMT	Operating line and number of stages by the McCabe/Thiele method
btdist	System of differential algebraic equations for a single-stage batch distillation
c5distBPeu	Calculation of a 5-component 16-stage distillation column
cpmlex	Membrane separation process using a complete mixing model
cpmdata	Specification of membrane data and operating conditions
crflex	Membrane separation process using a cross-flow model
crfdata	Specification of membrane data and operating conditions
crxf	Differential equations for a porous catalytic reaction
distBPeu	Multicomponent distillation calculation using the BP method
dyndist	Dynamics of a binary distillation process
drugf	System of differential equations for drug transfer
fugdist	Simple distillation calculation
mArea	Definition of an integral function required for the calculation of membrane area
mdif	Differential equation for a multicomponent diffusion system
multievapSI	Multiple-effect evaporation process (SI unit system)
phiHeu	Enthalpy of mixture and fugacity coefficient for each component
rdist	Balance equations at each stage
rxnf	Differential equations for an isothermal catalytic reaction
sinevap	Single-effect evaporation process
sldif	Differential equation for unsteady-state diffusion in a one-dimensional slab
spf	Differential equation for sphere diffusion
ssdist	Steady-state liquid composition at each stage
useabsf	Concentration in laminar downflow
usebtdist	Solution of the function btdist
usecrxf	Solution of the function crxf
usedrugf	Solution of the function drugf
usedyndist	Solution of the function dyndist
usemdif	Solution of the function mdif
userdist	Solution of the function rdist
userxnf	Concentration distribution and effectiveness factor
usesldif	Concentration in a one-dimensional slab
usespf	Solution of the function spf
usexaz	Solution of the function xaz
xaz	Differential equation for a binary mass transfer system

CHAPTER 8

airCooler	Preliminary design of an air cooler
bisecQL	Heat loss from a pipe flange
bisecQT	Heat transfer flux and temperature distribution
dpshell	Shell-side pressure drop
dptube	Tube-side pressure drop
ellipticPDE	Elliptic partial differential equation
funQL	Differential equation for flange heat transfer
funQT	Differential equation for heat transfer through wire
ht1D	One-dimensional heat transfer
htcoef	Calculation of the heat transfer coefficient
htcshell	Shell-side heat transfer coefficient
htctube	Tube-side heat transfer coefficient
hxcp	Tube-side and shell-side specific heat
hxndat	Specification of data for a shell-and-tube heat exchanger
hxnst	Calculation of shell-and-tube heat exchanger
hxrho	Tube-side and shell-side density
hxthc	Tube-side and shell-side heat conductivity
hxvis	Tube-side and shell-side viscosity
parab1D	One-dimensional parabolic PDE
parab2D	Two-dimensional parabolic PDE
parabDbc	Solution for constant boundary conditions
pipeTracer	Pipeline heat tracer
shellLMTD	Shell number and log-mean temperature difference
slab1T	Differential equation for one-dimensional heat transfer
slabMT	Differential equation for multiple slabs
tempPDE	Solution of the Laplace equation using the finite difference method

CHAPTER 9

bodezeta	Bode diagram as a function of ζ
csthcont	Differential equation for a heating process
csthcontp	Differential equation for a heating process with P-control
delaybode	Bode diagram of a 3rd-order process with time delay
highresp	Step response of a higher-order process
levelcon	Feedback control system for a 1st-order process
rootloc	Root locus
secpro	Control of a 2nd-order process using P-controller
sin1st	Sine response of a 1st-order process
step1	Step response
step2ndpro	Step response of a 2nd-order process
tdelay	Time-delay calculation
tunpidTd	Change derivative time
tunpidTi	Change integral time
usecsthcont	Solution of the function <code>csthcont</code>
usecsthcontp	Solution of the function <code>csthcontp</code>

CHAPTER 10

barnslp	Interior point method by Barnes' algorithm
bnbopt	Branch and bound algorithm
brentopt	Brent search algorithm
cgopt	Conjugate gradient algorithm
cycopt	Cyclic coordinate search method
dfpopt	David–Fletcher–Powell (DFP) method
fbnopt	Fibonacci search algorithm
gaopt	Genetic algorithm
grgopt	Generalized reduced gradient method
gsopt	Golden section search algorithm
hjopt	Hooke and Jeeves pattern search method
newtonopt	Newton's method
nmopt	Nelder and Mead's simplex method
ozopt	Zero–one programming method
rbopt	Rosenbrock's method
rosencopt	Rosen's gradient projection
saopt	Simulated annealing method
sdopt	Steepest descent algorithm
shubertopt	Shubert–Piyavskii algorithm
simplexlp	Two-phase simplex method
sqpopt	Sequential quadratic programming method
zoutopt	Zoutendijk's feasible direction method

Index

A

absbz function, 358–359
absf function, 347
Absolute humidity
 definition, 114
 interpolation, 68–69
 MATLAB® program, 114
Absorption, 356–359
Activity coefficient
 benzene/acetic acid system, 176–178
 expression, 176
 group contribution method
 UNIFAC method, 181–185
 UNIQUAC method, 180–181
 van Laar equation, 179
 Wilson equation, 179
adbpf function, 313
adfun function, 291
Adiabatic reaction, 311–314
airCooler function, 454
Air coolers
 calculation procedure, 454
 design calculations, 453
 MATLAB® program, 454–455
Andrade correlation, 119
Antoine equation, of vapor pressure,
 55–57, 132, 173
Arithmetic operators, 5
Arrhenius equation, of reaction rate
 constant, 269
Awoseyin method, 145

B

Backslash operator, 37
Backward difference formula, 70
barnslp function, 526
Batch reactor
 reaction parameters, 284–285
 semibatch reactors, 285–287
Benedict–Webb–Rubin (BWR) equation, 153
Benzene absorption, 358–359
Bernoulli equation, 234
BFGS method, *see* Broyden–Fletcher–Goldfarb–Shanno
 method
Binary distillation
 distillation column dynamics, 366–367
 ideal, 363–366
 MATLAB® program, 362–363, 365–367
 McCabe/Thiele method, 360–363
bindistMT function, 362
binVLE function, 191
Bioreactors, 308–311
bisecQT function, 414
Bisection method, 41
Block diagrams, 462–463
bnbopt function, 570

Bode diagram
 definition, 488
 (second)2nd-order process, 488
 time delay, 488–489
Boltzmann’s probability distribution function, 560
BP method, *see* Bubble-point method
Branch and bound method
 MATLAB® program, 570–575
 process steps, 569–570
brentopt function, 503
Brent’s quadratic fit method
 MATLAB® program, 503–505
 process steps, 502–503
Broyden–Fletcher–Goldfarb–Shanno (BFGS) method, 517
btdist function, 393
Bubble-point calculations, 204
Bubble-point (BP) method, 382–390
bubbleT function, 197, 199
bubbp function, 186–187
Built-in functions, 22–23
 constrained optimization, 577–579
 linear programming, 579–580
 mixed-integer programming, 580–581
 unconstrained optimization, 575–577
bvp4c function, 84
BWReq function, 154
BWR equation, *see* Benedict–Webb–Rubin equation
bzacfun function, 177
bzacP function, 177–178

C

Catalytic reaction model, 58–60
Catalytic reactors
 gas oil cracking reaction, 303–304
 packed bed reactor
 complex gas-phase reactions, 298–300
 conversion, 300–302
 gas-phase catalytic reaction, 297–298
 reaction parameters, 294–297
 straight-through transport reactor, 302–303
 Cauchy conditions, 88, 417
c5distBPeu function, 388–389
Central difference formula, 70
cgopt function, 513
Chang correlation, 369
Chemical equilibrium, 272–273
Chemical reactors
 batch reactor
 reaction parameters, 284–285
 semibatch reactors, 285–287
 bioreactors, 308–311
 catalytic reactors
 gas oil cracking reaction, 303–304
 packed bed reactor, 297–302
 reaction parameters, 294–297
 straight-through transport reactor, 302–303

- CSTR, 277–284
 design and analysis, 269
 membrane reactors, 304–308
 nonisothermal reactions
 adiabatic reaction, 311–314
 n-butane isomerization reaction, 316–319
 steady-state reaction, 314–316
 plug-flow reactor
 acetone cracking reaction, 290–294
 gas-phase decomposition reaction, 288–290
 mass balance, 287–288
 reaction rates
 chemical equilibrium, 272–273
 rate constant, 269–272
 reaction conversion, 273–275
 series reactions, 275–276
 Clausius–Clapeyron equation, 139–140
clear command, 16
 Closed-loop feedback control system
 block diagram, 474
 regulator problem, 474
 servo problem, 474
 step responses
 PI controller, 477
 PID controller, 478–479
 proportional controller, 475–477
cltank function, 244
 Colebrook equation, 43
 Command Window
 entering commands, 1–3
 exit command, 4
 help command, 3–4
compflow function, 253
compID function, 115–116, 119–120
 Complete-mixing model, 395–397
 Complex number, 13–14
 Complex process control, *see* Higher-order process control
 Compressibility factors, of natural gases
 expression, 145–146
 MATLAB® program, 146
 Compressible fluid flow
 critical flow, 250
 isothermal flow, 250, 252
 Computational limitations and constants, 16–17
compVPeqn function, 175
condG function, 126
condL function, 125
 Conjugate gradient method
 MATLAB® program, 513–516
 quadratic function minimization, 512–513
 Constrained optimization
 built-in functions, 577–579
 GRG method, 537–544
 Rosen’s gradient projection method, 528–532
 SQP method, 544–549
 Zoutendijk’s feasible direction method, 532–537
 Continuous stirred tank heater control, 480–484
 Continuous-stirred tank reactor (CSTR)
 exothermic reaction, 278–282
 multiple reactions, 282–284
 series CSTR, 277–278
corstsg function, 131
costald function, 118
 COSTALD method, 117–118
cpmlEx function, 396–397
crflex function, 398–399
 Cross-flow model, 397–399
crxf function, 341
csthcont function, 482–484
cstrmult function, 283
cubicEOSZ function, 155
 Cubic equation of state
 expression, 155
 fugacity coefficient, 171
 MATLAB® program, 156–157
 n-butane, molar volume of, 157
 parameters, 155–156
 Cubic spline interpolation, 64–65
cumtrapz function, 72–73
 Cyclic coordinate method
 MATLAB® program, 549–551
 process steps, 549
cycopt function, 549
- D**
- Davidon–Fletcher–Powell (DFP) method, 517
 Definite integral, 72–74
delaybode function, 488–489
delH function, 158
delHS function, 162
denL function, 116
 Density, of saturated liquids
 expression, 115
 MATLAB® program, 115–118
 Departure function, 158–164
deptfun function, 160
 Dew point
 calculations, 204
 interpolation of, 68–69
dewpt function, 187–188
 DFP method, *see* Davidon–Fletcher–Powell method
dfpopt function, 517
 Differential distillation, 390–394
 Differentiation, 70–72
diff function, 70–71
 Diffusion
 steady-state, 329–342
 unsteady-state, 343–348
 Diffusion coefficient
 gas phase, 144–145
 liquid phase, 143–144
 Direct search methods
 cyclic coordinate method, 549–551
 GA, 562–566
 Hooke and Jeeves pattern method, 551–553
 Nelder and Mead’s simplex method, 556–559
 Rosenbrock’s method, 554–556
 SA method, 559–562
 Dirichlet conditions, 88, 417
distBPeu function, 383
 Distillation
 binary, 360–367
 differential, 390–394
 multicomponent, 367–373
 rigorous steady-state, 373–390

- Double integral, 75
- Double-pipe finned-tube heat exchanger
- heat transfer coefficients and film coefficients, 447–448
 - MATLAB® program, 449–450
- dpshell* function, 440–442
- dptube* function, 440–441
- drugf* function, 336
- dyndist* function, 365
- E**
- Eduljee correlation, 369
- ellipticPDE* function, 424, 427
- Enclosed tank flow system, 243–245
- Enthalpy change, 158
- Enthalpy of mixture
- expression, 164–165
 - MATLAB® program, 165–167
- Enthalpy of vaporization
- Clausius–Clapeyron equation, 139–140
 - MATLAB® programs, 137–138, 140
 - Pitzer correlation, 138
 - Watson correlation, 137–138
- Equation of state
- cubic equation, 155–157
 - Lee–Kesler equation, 153–154
 - virial equation, 151–152
- Equilibrium conversion, 273–275
- Ergun equation, 261–262
- ethanrxn* function, 46–47
- Euler's method, 75–76
- Evaporation
- MATLAB® program, 350–351, 353–356
 - multiple-effect, 351–356
 - single-effect, 348–351
- exbco* function, 317
- expand* function, 18
- Extended Antoine equation, 132–133
- ezplot* function, 25, 28
- F**
- falp* function, 193
- False position method, 41
- fbnopt* function, 498
- Feedback control
- loop dynamics, 474–479
 - system stability, 484–485
- Fenske equation, 367–368
- Fibonacci method
- MATLAB® program, 498–499
 - minimization algorithm, 497–498
- Fick's law, 329, 338, 341
- 1st-order process control
- process dynamics, 464–466
 - Simulink® block diagram, 466–467
 - sinusoidal response, 467–468
 - step response, 466–468
- Fixed-point iteration method, 42
- flashBubP* function, 200, 202
- flashdrum* function, 194
- Flash evaporator, 192–193
- Fletcher–Reeves algorithm, 513
- Fluidized packed bed catalytic reactor, 82–83
- Fluid mechanics
- laminar flow
 - falling film, 217–219
 - falling particle, 219–220
 - horizontal annulus, 215–217
 - horizontal circular pipe, 214–215
 - Reynolds number, 213
 - non-Newtonian fluid flow
 - friction factor, 248–249
 - Reynolds number, 247–248
 - velocity profile, 245–247
 - packed bed fluid flow, 261–263
 - pipeline fluid flow
 - compressible flow, 250–252
 - equivalent pipe length, 231
 - excess head loss, 231
 - friction loss, 225–231
 - network, 237–242
 - overall pressure drop, 234
 - pipe size changes, 232
 - two-phase flow, 254–261
 - tank flow system
 - enclosed tank, 243–245
 - open tank, 242–243
- fminbnd* function, 69, 576
- fmincon* function, 577, 579
- fminimax* function, 577–579
- fminsearch* function, 69–70, 575–576
- fminunc* function, 575–576
- for* loop, 25, 32
- FORTRAN programming language, 91
- Forward difference formula, 70
- Fourier's law, 412
- Four-loop pipeline network, 240–242
- Frequency response analysis
- Bode diagram, 487–489
 - GM and PM, 492–493
 - Nichols chart, 487, 491
 - Nyquist diagram, 487, 490–491
- frfactor* function, 222
- Friction factor, 221–225
- fsolve* function, 272, 280, 375
- Fugacity coefficient
- pure species
 - acetylene gas, 170
 - expression, 168–169
 - MATLAB® program, 169–170
 - species in a mixture, 171–173
- fugdist* function, 372–373
- Fuller, Schettler, and Giddings's method, 144–145
- Function M-files, 21–23
- Functions
- built-in, 22–23
 - user-defined, 23–24
- fzero* function, 42
- G**
- GA, *see* Genetic algorithm
- Gain margin (GM), 492–493
- gaopt* function, 563, 566
- Gas-phase diffusion coefficients, 144–145
- Gas separation, *see* Membrane separation
- gdiffc* function, 144

geferm function, 310
 Generalized reduced gradient (GRG) method
 MATLAB® program, 539–544
 process steps, 537–539
 Genetic algorithm (GA)
 flowchart, 563
 MATLAB® program, 563–566
 process, 562
gfree function, 142
 Gibbs–Duhem equation, 176
 Gibbs free energy of formation
 definition, 142
 MATLAB® program, 142–143
 Gilliland correlation, 368–370
 GM, *see* Gain margin
 Golden section method
 MATLAB® program, 500–502
 process steps, 499–500
gradient function, 70–72
 Graphics
 ezplot function, 25, 28
 modification, 25–27
 multiple curves, 28–29
 plot function, 27–28
 three-dimensional plots, 29–30
 GRG method, *see* Generalized reduced gradient method
grgopt function, 539
 Group contribution method, 180–185
gsopt function, 500

H

Harg correlation, 369
 Harlecher–Braun equation, 174
hcapG function, 124
hcapL function, 122
hcRB function, 123
 Heat capacity
 gases
 MATLAB® program, 124–125
 polynomial correlation, 123
 liquids
 MATLAB® program, 122–123
 polynomial correlation, 121–122
 Rowlinson/Bondi method, 123
 Heat exchanger
 double-pipe, 447–450
 shell-and-tube, 429–447
 Heat of formation, of ideal gases
 expression, 140
 MATLAB® program, 140–141
 Heat transfer
 air coolers, 453–455
 heat exchanger, 429–450
 multidimensional heat conduction, 416–429
 one-dimensional heat conduction, 407–416
 tracers, 450–453
 Hengstebeck correlation, 368
 Hermitian transpose, of complex number, 14
 Hessian matrix, 510–511
 Heterogeneous catalytic reaction model, 58–60
hform function, 140

Higher-order process control
 lead/lag, 472
 time delay process, 472–474
 unit step response, 471–474
highresp function, 472
 Histogram, 29–30
hjopt function, 552
 Hoffmann–Florin equation, 135
 Hohman and Lockhart correlation, 369
 Hooke and Jeeves pattern method
 MATLAB® program, 552–553
 process steps, 551–552
H2OPT function, 106, 109
htcoef function, 449–450
htcshell function, 438–440
htctube function, 438–439
ht1D function, 408–409
humidest function, 114–115
 Humidity
 absolute, 114
 MATLAB® program, 114–115
 relative, 113
 Huntington friction factor, 256
hvapn function, 137
hxcp function, 437–438
hxndat function, 443
hxnst function, 443, 446–447
hxrho function, 437–438
hxthc function, 437–438
hxvis function, 437
hzanpipe function, 216
hzpipe function, 214

I

ibwegf function, 205–206
 Ideal binary distillation, 363–366
if statement, 24–25
impulse function, 466
inline function, 23
 Integration
 definite, 72–74
 multiple, 74–75
 Interior point method
 MATLAB® program, 526–528
 process steps, 526
 International Association for Properties of Water and Steam–Industrial Formulation 1997 (IAPWS-IF97), 103–104
interp1 function, 65
interp2 function, 67
 Interpolation
 cubic spline, 64–65
 multidimension, 67–69
 one-dimensional data, 65–67
 polyfit function, 62–63
 polynomials, 60–62
 three-dimensional, 68
 two-dimensional, 67–69
intlinprog function, 580–581
 Inverse Laplace transform, 459–460
 Isothermal flash calculations, 205
 Isothermal plug-flow reactor, 205

J*jacob* function, 511**K**

Karush–Kuhn–Tucker (KKT) conditions, 544–545
khmix function, 165
Kirkbride equation, 370
KKT conditions, *see* Karush–Kuhn–Tucker conditions

L

Lagrange interpolation method, 67
Laminar flow
 falling film, 217–219
 falling particle, 219–220
 horizontal annulus, 215–217
 horizontal circular pipe, 214–215
 Reynolds number, 213
Lang–Miller method, 239–240
Laplace transform, 459–460
Lee–Kesler equation
 1-butene, compressibility factor of, 154
 constants, 153
 expression, 153
 MATLAB® program, 154
levelcon function, 475
Liddle correlation, 368–369
Linear equation systems, 35–39
Linear programming (LP)
 built-in functions, 579–580
 formulation, 520–521
 interior point method, 526–528
 simplex method, 521–522
 two-phase simplex method, 522–526
Linear regression analysis, 50–51
linprog function, 579–580
linspace command, 4, 8
Liquid-phase diffusion coefficients
 estimation, 143–144
 MATLAB® program, 144
log function, 22
Loops, 24–25
LP, *see* Linear programming
lsim function, 466
lsqlin function, 577–578
lsqnonlin function, 576

M

Mach number, 250
mArea function, 398–399
margin function, 492
Mass transfer
 absorption, 356–359
 diffusion
 steady-state, 329–342
 unsteady-state, 343–348
 distillation
 binary, 360–367
 differential, 390–394

 multicomponent, 367–373
 rigorous steady-state, 373–390
 evaporation, 348–356
 membrane separation
 complete-mixing model, 395–397
 cross-flow model, 397–399
Matrices, 9–13
Maxwell–Stefan equation, 330
mbrmult function, 308
mcatb function, 304
McCabe/Thiele method, 360–363
McCormick correlation, 369
mdif function, 332
Membrane reactors, 304–307
 multiple reactions, 307–308
Membrane separation
 complete-mixing model, 395–397
 cross-flow model, 397–399
 MATLAB® program, 396–399
membrx function, 306
Method of lines, 418–423
M-files
 adding comments, 20–21
 function, 21–23
 script, 19–20, 32
Michaelis–Menten enzyme kinetics, 54–55
microrx function, 289–290
Minimum reflux ratio, 368
Mixed-integer programming
 branch and bound method, 569–575
 built-in functions, 580–581
 zero-one problems, 566–569
Modified Raoult’s law, 188–195
Molkavov correlation, 369
Monod equation, 308–309
Moore–Penrose pseudoinverse of matrix, 36
Muller’s method, 41–42
Multicomponent diffusion, 330–333
Multicomponent distillation
 feed stage location, 370
 Fenske equation, 367–368
 Gilliland correlation, 368–370
 MATLAB® program, 371–373
 number of stages, determination of, 370–371
 Underwood equation, 368
Multidimensional heat conduction
 method of lines, 418–423
 steady-state, 424–429
 unsteady-state, 416–418
Multidimensional interpolation, 67–69
multievap function, 353
multievapSI function, 353
Multilayer cylinders, heat transfer through, 410–411
Multilayered slab, heat transfer through
 expression, 409
 MATLAB® program, 410–412
Multiple-effect evaporators
 calculation procedure, 353
 MATLAB® program, 353–356, 591–592
 schematic diagram, 351
 simplified diagram, 352
Multiple integrals, 74–75

N

Nelder and Mead's simplex method
 MATLAB® program, 557–559
 process steps, 556–557
 Neumann conditions, 88, 417
newtonopt function, 511
 Newton's method
 MATLAB® program, 511–512
 for nonlinear systems, 41–42
 quadratic approximation, 510–511
ngasZ function, 146
 Nichols chart
 definition, 487
 (second)2nd-order process, 491
nmopt function, 557
nnDfun function, 249
nnhzpipe function, 246
 Nonisothermal reactions
 adiabatic reaction, 311–314
n-butane isomerization reaction, 316–319
 steady-state reaction, 314–316
 Nonlinear systems
 polynomial equations, 40–41
 zeros, 41–47
 Non-Newtonian fluid flow
 friction factor, 248–249
 Reynolds number, 247–248
 velocity profile, 245–247
 Numerical methods
 differentiation, 70–72
 integration, 72–75
 interpolation, 60–69
 linear systems, 35–39
 nonlinear systems, 40–47
 ODE, 75–87
 optimization, 69–70
 PDE, 87–91
 regression analysis, 47–60
 Nyquist diagram
 definition, 487
 MATLAB® program, 489–490
 (second)2nd-order process
 nyquist function, 489–490
 polar function, 489–490
 time delay, 490

O

ode45 function, 30, 276, 278, 298
 One-dimensional diffusion, 329–330
 One-dimensional heat transfer
 multilayer cylinders, 410–411
 multilayered slab, 409–412
 one-dimensional slab, 407–409
 wires, 413–415
 One-dimensional interpolation, 65–67
 One-dimensional parabolic PDE, 89–91
 One-dimensional slab, heat transfer in
 expression, 407
 MATLAB® program, 408–409
 Open tank flow system, 242–243
 Optimization
 built-in functions, 69–70

constrained

GRG method, 537–544
 Rosen's gradient projection method, 528–532
 SQP method, 544–549
 Zoutendijk's feasible direction method, 532–537
 direct search methods
 cyclic coordinate method, 549–551
 GA, 562–566
 Hooke and Jeeves pattern method, 551–553
 Nelder and Mead's simplex method, 556–559
 Rosenbrock's method, 554–556
 SA method, 559–562

LP method

formulation, 520–521
 interior point method, 526–528
 simplex method, 521–522
 two-phase simplex method, 522–526

MATLAB built-in functions, 575–581

mixed-integer programming

branch and bound method, 569–575
 zero–one problems, 566–569
 role, in chemical engineering, 497
 unconstrained

Brent's quadratic fit method, 502–505
 conjugate gradient method, 512–516
 Fibonacci method, 497–499
 golden section method, 499–502
 Newton's method, 510–512
 quasi-Newton method, 516–520
 Shubert–Piyavskii algorithm, 505–507
 steepest descent algorithm, 507–510

Ordinary differential equations (ODEs), 30–31

boundary-value problems, 84–87
 initial-value problems, 75–77
 stiff system, 77, 81

ozopt function, 567

P

PACER, 91

Packed bed fluid flow
 MATLAB® program, 262–263
 pressure drop, 261–262

Packed bed reactor

complex gas-phase reactions, 298–300
 conversion, 300–302
 gas-phase catalytic reaction, 297–298

parabDbc function, 419

parab1D function, 418–419

parab2D function, 418–419

Partial differential equations (PDEs)

classification, 87–88
 one-dimensional temperature profile, 89–91
pdepe function, 88–89

Partial fraction expansion, 460–461

pbconv function, 301

pbrmf function, 297

pbrmult function, 299

pbrpar function, 296

pdepe function, 88–89

PDEs, *see* Partial differential equations

Peng–Robinson (PR) equation, 155, 160, 168

Phase margin (PM), 492–493

phigas function, 169

- phiHeu* function, 385
phimix function, 172
- Physical properties
 compressibility factor, 145–147
 density, 115–118
 diffusion coefficient
 gas phase, 144–145
 liquid phase, 143–144
 enthalpy of vaporization, 137–140
 Gibbs free energy of formation, 142–143
 heat capacity
 gases, 123–125
 liquids, 121–123
 heat of formation, 140–142
 humidity, 113–115
 surface tension, 128–131
 thermal conductivity
 gases, 126–128
 liquids, 125–126
 vapor pressure, 132–136
 viscosity
 gases, 120–121
 liquids, 118–119
- pinv* function, 36
- Pipe flange, heat loss through
 expression, 415
 MATLAB® program, 416
- Pipeline fluid flow
 compressible flow
 critical flow, 250
 isothermal flow, 250, 252
 equivalent pipe length, 231
 excess head loss, 231
 friction loss, 225–231
 network, 237–242
 overall pressure drop, 234
 pipe size changes, 232
 two-phase flow, 254–261
- Pipelines, heat tracing in
 calculation procedure, 451
 MATLAB® program, 451–453
 resistances, 450–451
- pipeTracer* function, 451–452
- pipnet* function, 241
- Pitzer correlation, 138
- pitzersg* function, 131
- plot* function, 27–28
- plot3* function, 29
- Plug-flow reactor
 acetone cracking reaction, 290–294
 gas-phase decomposition reaction, 288–290
 mass balance, 287–288
- PM, *see* Phase margin
- pmod* function, 31
- polyfit* function, 52, 62–63
- Polynomial interpolation, 60–62
- Polynomial regression analysis, 52–53
- Population growth model, 31–32
- Powell's function, 551, 559
- Pressure–temperature (P–T) diagram, 103–104
- Process control
 block diagrams, 462–463
 continuous stirred tank heater, 480–484
 feedback control loops, 474–479
- 1st-order process, 464–468
- frequency response analysis
 Bode diagram, 487–489
 GM and PM, 492–493
 Nichols chart, 487, 491
 Nyquist diagram, 487, 490–491
- higher-order process, 471–474
- Laplace transform, 459–460
- (second)2nd-order process, 468–471
- partial fraction expansion, 460–461
- state-space representation, 463–464
- Process system engineering software, 91–93
- Proportional controller, 475–477
- Proportional–integral (PI) controller, 477
- Proportional–integral–derivative (PID) controller, 478–479
- prVp* function, 132
- Q**
- quad* function, 73
quadprog function, 578–579
- Quasi-Newton method
 MATLAB® program, 517–520
 process steps, 516–517
- R**
- Random Fibonacci sequence, 32–33
- Raoult's law, 185–188
- Rarey/Moller equation, 135–136
- Rate constant, 269–272
- rateconst* function, 270
- rbopt* function, 554
- rdist* function, 375–376
- Reaction conversion, 273–275
- Reaction rates
 chemical equilibrium, 272–273
 rate constant, 269–272
 reaction conversion, 273–275
 series reactions, 275–276
- Redlich–Kwong (RK) equation, 155, 159, 168
- Reflux ratio, 360
- reorder* function, 271
- Regression analysis
 elementary statistics, 47–49
 linear, 50–51
 polynomial, 52–53
 random number generation, 50
- Relative humidity, 113
- Reynolds number
 laminar flow, 213
 non-Newtonian fluid flow, 247–248
- Riedel equation, 174
- Rigorous steady-state distillation
 BP method, 382–390
 MATLAB® program, 375–376, 383–389
 MESH equations, 376–379
 three-stage distillation column, 374–376
 tridiagonal matrix method, 379–381
- rlocus* function, 485
- Robbins conditions, 88, 417
- Root locus diagram, 485–486
- roots* function, 40

- Rosenbrock's method
 MATLAB® program, 554–556
 process steps, 554
- rosencpt* function, 529
- Rosen's gradient projection method
 MATLAB® program, 529–532
 minimization problem, 528–529
- Roulette wheel selection, 563
- Rowlinson/Bondi method, 123
- Runge–Kutta method, 76
- rxnf* function, 339
- rxnfun* function, 46
- S**
- SA method, *see* Simulated annealing method
- satsteam* function, 111–112
- Saturated steam
 basic equations, 104–106
 MATLAB® program, 111–113, 587–590
 parameters and auxiliary equations, 104
 property equations, 110–111
- Script M-file, 19–20, 32
- sdopt* function, 508
- Secant method, 41
- 2nd-order polynomial interpolation, 61–62
- 2nd-order process control
 MATLAB® program, 470
- Nyquist diagram
 nyquist function, 489–490
 polar function, 489–490
- process dynamics, 468–470
- Simulink® block diagram, 471
 step response, 470–471
- secpro* function, 475–476
- Semibatch reactors, 285–287
- semibrx* function, 287
- Sequential quadratic programming (SQP) method
 KKT conditions, 544–545
 MATLAB® program, 546–549
- Series reactions
 concentration profiles, 276
 MATLAB® program, 276
 reaction rates, 275
- serrb* function, 276
- Shacham equation (SC), 221
- Shell-and-tube heat exchanger
 calculation procedure, 436–437
 heat transfer coefficient, 432–435
 log-mean temperature difference, 429–430
 MATLAB® program, 430–432, 437–447
 pressure drop, 435–436
- shellLMTD* function, 430, 432
- shubertopt* function, 506
- Shubert–Piyavskii method
 MATLAB® program, 506–507
 process steps, 505–506
- simplexlp* function, 523
- Simplex method, 521–522
- Simpson's rule, 72
- Simulated annealing (SA) method
 MATLAB® program, 560–562
 process, 559–560
- Simulink® block diagram, 466–467, 471, 477
- sinevap* function, 350
- Single-effect evaporators, 348–351
- Single-stage batch distillation, 393
- sinx* function, 21–22
- slabmT* function, 412
- slab1T* function, 408
- sldif* function, 344
- Smoker equation, 370–371
- smopt* function, 560
- Soave–Redlich–Kwong (SRK) equation, 43–44, 155, 160, 168
- SPEEDUP (Simulation Program for the Economic Evaluation and Design of Unsteady-State Processes), 91
- SQP method, *see* Sequential quadratic programming method
- sqpopt* function, 546
- ssdist* function, 366
- State-space representation, 463–464
- Steady-state diffusion
 isothermal catalyst particles, 337–342
 MATLAB® program, 331–332, 335–337, 339–342
 multicomponent diffusion, 330–333
 one-dimensional diffusion, 329–330
 solid sphere, 333–335
- Steady-state heat conduction
 expression, 424
 MATLAB® program, 424–429
 two-dimensional elliptic equations, 426–429
- Steady-state nonisothermal reactions, 314–316
- Steepest descent method
 MATLAB® program, 508–510
 minimization problem, 507–508
- step2ndpro* function, 470
- Stiff differential equations, 77, 81
- Stirred-tank heating process control
 closed-loop response, 482, 484
 controller output equation, 480–481
 MATLAB® program, 482–484
 open-loop response, 483
- Surface tension, of liquids
 expression, 128–129
 MATLAB® program, 129–131
- surfL* function, 129
- Symbolic operations
 creation, 17–18
 substitution, 18–19
- sym* function, 17–18
- syms* function, 17–18
- T**
- tdelay* function, 473
- tempPDE* function, 427–428
- Thermal conductivity
 gases
 expression, 126
 MATLAB® program, 126–128
- liquids
 expression, 125
 MATLAB® program, 125–126
- Thermodynamic properties
 saturated steam, 110–113
 water, 106–109

- Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use, revised IAPWS Formulation 1995 for, 110
- Thermodynamics
- activity coefficient
 - benzene/acetic acid system, 176–178
 - expression, 176
 - group contribution method, 180–185
 - van Laar equation, 179
 - Wilson equation, 179
 - equation of state
 - cubic equation, 155–157
 - Lee–Kesler equation, 153–154
 - virial equation, 151–152
 - fluid properties
 - departure function, 158–164
 - enthalpy change, 158
 - enthalpy of mixture, 164–168
 - fugacity coefficient
 - pure species, 168–170
 - species in a mixture, 171–173
 - vapor–liquid equilibrium
 - modified Raoult’s law, 188–195
 - Raoult’s law, 185–188
 - ratio of fugacity coefficients, 195–202
 - vapor–liquid–liquid equilibrium
 - bubble-point calculations, 204
 - dew-point calculations, 204
 - isothermal flash calculations, 205
 - MATLAB® program, 205–207
 - vapor pressure
 - Antoine equation, 173
 - Harlecher–Braun equation, 174
 - MATLAB® program, 175
 - Riedel equation, 174
 - Three-dimensional interpolation, 68
 - Three-dimensional (3D) object generation, 33–34
 - Three-dimensional plots, 29–30
 - Three-dimensional unsteady-state heat conduction equation, 416
 - Transfer function, representation of, 461–462
 - Trapezoidal rule, 72
 - trapz* function, 72–73
 - Tridiagonal matrix method, 379–381
 - tunpid* function, 478–479
 - twinobj.m* script, 33
 - Two-dimensional interpolation, 67–69
 - Two-dimensional unsteady-state heat conduction equation, 417
 - Two-phase simplex method
 - MATLAB® program, 523–526
 - process steps, 522–523
 - twophdP* function, 258
 - twophreg* function, 255
 - Two-point boundary-value problems, 84–85
- U**
- Ultimate gain, 492
 - Unconstrained optimization
 - Brent’s quadratic fit method, 502–505
 - built-in functions, 575–577
 - conjugate gradient method, 512–516
 - Fibonacci method, 497–499
- golden section method, 499–502
- Newton’s method, 510–512
- quasi-Newton method, 516–520
- Shubert–Piyavskii algorithm, 505–507
- steepest descent algorithm, 507–510
- Underwood equation, 368
- UNIFAC (Universal Quasi-Chemical Functional Group Activity Coefficient), 181–185
- unifgam* function, 181
- UNIQUAC (Universal Quasi-Chemical Model), 180–181
- Unsteady-state diffusion
 - falling laminar film, 345–348
 - MATLAB® program, 344–345, 347–348
 - one-dimensional slab, 343–345
- Unsteady-state heat conduction
 - initial and boundary conditions, 417–418
 - MATLAB® program, 419–423
 - three-dimensional equation, 416
 - two-dimensional equation, 417
- useabsf* function, 347–348
- useadbpf* function, 314
- usebtdist* function, 393–394
- usecltank* function, 244–245
- usecrxf* function, 342
- usecsthcontp* function, 482–484
- usedrugf* function, 337
- usedyndist* function, 366–367
- useexbco* function, 317, 319
- usegeferm* function, 311
- usenbrmult* function, 308
- usemdif* function, 332–333
- usemembrx* function, 306–307
- usemicrorx* function, 290
- usepbconv* function, 301–302
- usepbmult* function, 299–300
- usepiperf* function, 242
- User-defined functions, 23–24
- userdist* function, 375–376
- userxnf* function, 340
- usesecpro* function, 476–477
- usesemibrx* function, 287
- usesldif* function, 344–345
- usespf* function, 335
- useunifgam* function, 183
- useunifgam2* function, 184
- usexaz* function, 331
- V**
- van der Pol equation, 78–79
 - van der Waals (VDW) equation, 155, 159, 168, 171
 - van Laar equation, 179
 - Van Winkle and Todd correlation, 369
 - Vapor–liquid equilibrium
 - modified Raoult’s law, 188–195
 - Raoult’s law, 185–188
 - ratio of fugacity coefficients, 195–202
 - Vapor–liquid–liquid equilibrium
 - bubble-point calculations, 204
 - dew-point calculations, 204
 - isothermal flash calculations, 205
 - MATLAB® program, 205–207
 - Vapor–liquid two-phase vertical downflow, 260

- Vapor pressure
 Antoine equation, 132, 173
 Harlecher–Braun equation, 174
 MATLAB® program, 132–134, 136, 175
 Riedel equation, 174
- VDW equation, *see* van der Waals equation
- Vectors, 6–9
- virialEOS* function, 152
- Virial equation of state
 departure functions, 159
 ethane, compressibility factor and molar volume of, 152
 expression, 151–152
 fugacity coefficient, 168, 171
 MATLAB® program, 152
- Viscosity
 gases
 expression, 120
 MATLAB® program, 120–121
 of liquid propane, 57–58
 liquids
 expression, 118–119
 MATLAB® program, 119
- visG* function, 120
- visL* function, 119
- Volume-swept three-dimensional (3D) object generation, 33
- vpRM* function, 136
- vpwagner* function, 134
- vtfun* function, 220
- vtwall* function, 218
- W**
- Wagner equation, 134–135
- Water–gas shift reaction, 272–273
- Water–isobutanol equilibrium calculations, 205–207
- Water properties
 basic equations, 104–106
 MATLAB® program, 106–109, 587–590
 parameters and auxiliary equations, 104
- Watson correlation, 137–138
- Well-mixed tanks, 79–80
- while* loop, 25
- whos* command, 17
- wilact* function, 179
- Wilke–Chang method, 143
- Wilson equation
 expression, 179
 MATLAB® program, 179–180
- Wire surface, heat transfer in
 differential volume, 413
 expression, 413
 heat flux and temperature profile, 414
 MATLAB® program, 414
- X**
- xaz* function, 331
- Z**
- Zero–one programming method
 MATLAB® program, 567–569
 minimization problem, 566
- zLK* function, 153–154
- Zoutendijk’s feasible direction method
 MATLAB® program, 533–537
 minimization problem, 532–533
- zoutopt* function, 533