Liverpool John Moores University
Control Systems Research Group
School of Engineering

Optimal Design of Controllers for Multivariable Processes Using Genetic Algorithms

Constantinos Vlachos

A thesis submitted in partial fulfilment of the requirements of Liverpool John Moores University for the degree of Doctor of Philosophy (Ph.D.)

This research project was carried out in collaboration with British Nuclear Fuels Limited

April 2000

In any particular theory there is only as much real science as there is mathematics.

IMMANUEL KANT

The most beautiful thing we can experience is the mysterious. It is the source of all true art and all science. He to whom this emotion is a stranger, who can no longer pause to wonder and stand rapt in awe, is as good as dead: his eyes are closed.

ALBERT EINSTEIN

To my parents

Στους γονείς μου

Abstract

This thesis focuses on the development and analysis of general methods for the optimal design of controllers for multivariable processes in a numerical optimisation framework, where genetic algorithms (GAs) are used to optimise a number of specially formulated objective functions. Strong emphasis is given on the generality and open architecture of the proposed methods, which are shown to be applicable to a wide range of real-world multivariable control problems.

A novel objective function is proposed for single-input, single-output (SISO) processes, that enables the designer to explicitly specify the performance specifications associated with a given problem in terms of time-domain bounds on the closed-loop responses. The proposed objective function is then experimentally investigated using a simple two-term parametric controller tuning problem. The obtained results are analysed and compared with those obtained using a number of popular conventional objective functions, as well as using a number of standard controller tuning methods. It is shown that the proposed objective function is inherently capable of accurately quantifying complex performance specifications in the time domain, which cannot normally be employed in conventional controller design/tuning methods. Finally, the proposed objective function is generalised to treat multi-input, multi-output (MIMO) control problems.

The proposed objective functions mentioned above permit the existence of a (usually infinite) set of multiple optimal solutions. To enable GAs to efficiently identify multiple optimal solutions, standard GAs are extended using a novel fitness assignment strategy called adaptive fitness sharing, which is based on the techniques of niche formation and speciation. It is shown that the proposed method enables the GA to evolve a population whose members are almost uniformly distributed within the optimal solution set. It is also shown that adaptive fitness sharing consistently outperforms the proportionate and rank-based fitness assignment strategies, in terms of both performance (higher degree of achieved uniformity), and robustness (less sensitivity to initial conditions). The time complexity of adaptive fitness sharing is shown to be better than that of conventional fitness sharing. Finally, adaptive fitness sharing is shown to be capable of adapting the phenotypic density of the population as required, while maintaining a high degree of uniformity throughout the search run. This facilitates the application of adaptive fitness sharing to on-line optimisation problems.

A new method for the automatic tuning of decentralised PI controllers for multivariable processes is proposed, based on GAs and the MIMO objective function developed earlier in this work. GAs are employed for the minimisation of this function, and the method of adaptive fitness sharing is used, in order to maximise the diversity of the obtained family of optimal solutions. The proposed tuning method is shown to be directly applicable to the automatic tuning of a wide range of linear or non-linear multivariable controllers, and

not just PI controllers. Simulation results are presented to illustrate the effectiveness of the proposed method. It is shown that the method enables the identification of a diverse family of controllers that completely satisfy the specifications, if such controllers exist. The designer can then manually examine the obtained family of controllers, and choose one which also satisfies additional qualitative optimality criteria which cannot easily be expressed in mathematical terms. The suitability of GAs in this optimisation problem is supported by statistically comparing them with two conventional optimisation methods, where it is shown that GAs have higher success rates and are more immune to noise.

A new solution to the Shell standard control problem is presented, based on GAs. The proposed control scheme includes two linear discrete-time PID controllers with integral anti-windup and a multivariable Smith predictor to provide the required process output regulation, while the process input minimisation problem is analytically solved on-line, by estimating the two unmeasured disturbances entering the Shell process and solving the associated linear program. GAs are successfully applied to the automatic tuning of the two PID controllers according to the given specifications, using an extension of the MIMO objective function developed earlier in this work. Extensive simulation results are presented, which show that the proposed control scheme is robustly stable and that its robust performance is comparable to that of more computationally intensive approaches, such as the Quadratic Dynamic Matrix Control (QDMC) and other algorithms. Making the specifications more conservative is shown to improve the robustness of the scheme in the face of model uncertainties. It is also demonstrated, through simulation, that the proposed control scheme satisfies all steady-state specifications for all five prototype test cases published by Shell, which include the nominal process model as well as a number of worst-case uncertain models. Finally, the success of the proposed GA-based control scheme demonstrates the potential of the application of GAs and the objective functions developed in this work, to large-scale multivariable process control problems.

Contents

Abstract *iv*
Acknowledgements *x*
Glossary of Symbols *xi*

1 Introduction – Survey and Thesis Outline 1

1.1 Introduction 1
1.2 Optimisation and Control Systems 1
1.2.1 Advantages of Optimisation-Based Controller Design 3
1.2.2 Limitations of Optimisation-Based Controller Design 4
1.3 Genetic Algorithms 4
1.4 Genetic Algorithms and Control Systems 7
1.4.1 Genetic Algorithms in Process Modelling 7
1.4.2 Genetic Algorithms in Process Control 8
1.5 Project Scope 10
1.6 Thesis Outline 11
1.7 Summary 13

2 Genetic Algorithms and Function Optimisation 14

2.1 Introduction 14
2.2 Typical Search Spaces in Function Optimisation 14
2.3 Conventional Optimisation Algorithms 16
2.4 Overview of Genetic Algorithms 18
2.4.1 Reproduction 19
2.4.2 Crossover 20
2.4.3 Mutation 20
2.4.4 Generation Gap and Elitism 21
2.4.5 Structure of the Simple Genetic Algorithm 21
2.5 Mathematical Foundations of Genetic Algorithms 22
2.5.1 Similarity Templates (Schemata) 23
2.5.2 The Fundamental Theorem of Genetic Algorithms 24
2.6 Genetic Algorithm Implementation Issues 30

2.6.1 Improving the Reproduction Operator 30
2.6.2 Selection of String Encoding 32
2.6.3 Selection of Genetic Algorithm Parameters 35
2.6.4 Incorporating Problem Specific Knowledge 37
2.6.5 Genetic Algorithm Termination 38
2.7 Multiobjective Optimisation and Genetic Algorithms 38
2.8 Genetic Algorithms as Universal Optimisers 43
2.9 Summary 43

3 Analysis and Design of Objective Functions for Control Systems 45

3.1 Introduction 45
3.2 Conventional Objective Functions 45
3.2.1 Limitations of Conventional Objective Functions 47
3.3 A Novel Objective Function for Control Systems 48
3.3.1 Objective Function Formulation 49
3.3.2 Optimal and Strictly Optimal Solutions 51
3.4 Experimental Analysis 52
3.4.1 The Performance Specifications 53
3.4.2 Conventional Objective Functions 54
3.4.3 PI Controller Tuning Methods 55
3.4.4 The Proposed Objective Function 56
3.4.5 Comments and Discussion 65
3.5 Generalisation to Multivariable Systems 66
3.6 Summary 69

4 Locating Multiple Optimal Solutions Using Genetic Algorithms 71

4.1 Introduction 71
4.2 Multiple Optimal Solutions and Genetic Drift 71
4.3 Niche Formation and Speciation in Genetic Algorithms 73
4.3.1 Fitness Sharing 75
4.4 The Proposed Niche Induction Method 77
4.4.1 Proportionate Fitness Assignment 77
4.4.2 Population Ranking 78
4.4.3 Adaptive Fitness Sharing 80
4.4.4 Sharing Functions 84
4.4.5 Optimal Sharing Radius Calculation 86
4.5 Experimental Analysis 88
4.5.1 Population Diversity Measure 89
4.5.2 Fitness Assignment Configurations 91
4.5.3 Simulation Results – No Mutation 92

4.5.4 Simulation Results – Effects of Mutation 97
4.5.5 Simulation Results – Statistical Tests 101
4.5.6 Effects of Parameter Scaling 104
4.5.7 Adaptation of the Population Density 106
4.5.8 Simulation Results in the Time Domain 108
4.6 Summary 109

5 Decentralised PI Controller Tuning for Multivariable Processes – A Genetic Approach 111

5.1 Introduction 111
5.2 Limitations of Existing PI/PID Tuning Methods 111
5.3 The Proposed Decentralised PI Controller Tuning Method 112
5.3.1 Decentralised PI Control of Multivariable Processes 113
5.3.2 Optimisation Problem Formulation 114
5.3.3 Boundary Functions and Set Point Test Patterns 115
5.4 Decentralised PI Controller Tuning Examples 117
5.4.1 PI Controller Tuning Example 1 117
5.4.2 PI Controller Tuning Example 2 117
5.4.3 PI Controller Tuning Example 3 118
5.4.4 Closed-Loop System Formulation 118
5.4.5 Performance Specifications 119
5.4.6 Genetic Algorithm Configuration 120
5.5 Simulation Results 121
5.5.1 Tuning Example 1 121
5.5.2 Tuning Example 2 123
5.5.3 Tuning Example 3 125
5.5.4 Search Landscape Complexity 128
5.5.5 Comparison with the Relay Feedback Technique 131
5.6 Comparison with Conventional Optimisation Methods 133
5.6.1 Computation Time Comparisons 134
5.7 Summary 135

6 Solution to the Shell Standard Control Problem Using Genetically Tuned PID Controllers 136

6.1 Introduction 136
6.2 The Shell Standard Control Problem 136
6.3 Design Methodology 138
6.3.1 The Output Regulation Problem 138
6.3.2 The Input Minimisation Problem 141
6.3.3 LP Solution Feasibility Analysis 143

6.3.4 Unmeasured Disturbance Estimation 146
6.3.5 Multivariable Smith Predictor Control 149
6.3.6 The Closed-Loop System 151
6.4 PID Controller Tuning 152
6.4.1 Objective Function Formulation 153
6.4.2 Application to the Shell Standard Control Problem 154
6.4.3 Genetic Algorithm Configuration 156
6.5 Simulation Results 157
6.5.1 Measurement Noise and Disturbance Variations 161
6.5.2 Robust Stability and Performance Tests 166
6.5.3 Genetic Algorithm Convergence 172
6.5.4 Search Landscape Complexity 174
6.6 Summary 178

7 Conclusions – Main Contributions and Further Work 180

7.1 Introduction 180
7.2 Summary of Main Contributions 180
7.2.1 Novel Objective Function Formulation for Control Systems 180
7.2.2 Adaptive Fitness Sharing – An Extension of the Standard GA 181
7.2.3 Parametric Controller Tuning for Multivariable Processes 182
7.2.4 Solution to the Shell Standard Control Problem 183
7.3 Recommendations for Further Work 183
7.3.1 Improved Objective Functions for Control Systems 183
7.3.2 Multiobjective Optimisation 184
7.3.3 Real-Time and Adaptive Control 184
7.4 Summary 185

A The Shell Standard Control Problem 186

B Minimal State-Space Realisations 191

References & Bibliography 195
Published Work 207

Acknowledgements

I would like to thank my supervisors, Prof. David Williams and Dr. Barry Gomm, for their support and encouragement throughout this research investigation. Their fruitful interactions and feedback helped transform the initial idea into reality, and made this project very interesting and exciting. Many thanks also go to my early supervisor, Dr. John Evans, for his support and encouragement during the early stages of this work. The support of Liverpool John Moores University Research Fund and British Nuclear Fuels Limited, who funded this research project, is also gratefully acknowledged.

It is my privilege to express my gratitude to my cousin and dearest friend Athanasios Spiropoulos, for introducing me to the wonderful world of mathematics. Undoubtedly, this project has been greatly influenced by our numerous discussions on this subject.

Finally, I reserve my very deepest appreciation for my family. It is a hackneyed theme to thank loved ones for patience and understanding while a project is being undertaken, but now I know why, and do give heartfelt thanks.

Constantinos Vlachos
Liverpool, UK, April 2000

cvlachos@mail.com
c.vlachos@livjm.ac.uk
cvlachos@hotmail.com

Trademarks

MATLAB® is a registered trademark of The MathWorks, Inc.
SIMULINK® is a registered trademark of The MathWorks, Inc.
Pentium® III is a registered trademark of Intel Corporation.

Glossary of Symbols

$\mathbb{N}$	Set of natural numbers, $\mathbb{N} = \{0,1,2,\ldots\}$
$\mathbb{R}$	Field of real numbers
$\mathbb{R}_+$	Set of non-negative real numbers
$\mathbb{R}^n$	n-dimensional vector space over $\mathbb{R}$
$\mathcal{C}$	Unit hypercube used in adaptive fitness sharing
$\mathcal{D}$	Set of permissible solutions (controllers)
$\tilde{\mathcal{D}}$	Set of strictly optimal solutions (controllers)
$\tilde{\mathcal{D}}_{\mathbf{h}}$	Set $\tilde{\mathcal{D}}$ when scaled using vector $\mathbf{h}$
$\mathcal{H}$	Hypercuboid of size $\mathbf{h}$ used in adaptive fitness sharing
$\mathcal{S}$	Time-domain performance specifications
$\mathbf{0}$	Zero vector, zero matrix
$\mathbf{A}$	State matrix of state-space model
$\mathbf{B}$	Input matrix of state-space model
b	Hypervolume ratio between $\tilde{\mathcal{D}}_{\mathbf{h}}$ and $\mathcal{C}$
$\mathbf{C}$	Output matrix of state-space model
c_i, c_{ss}	Time-domain performance specification coefficients
D, $\mathbf{D}$	General controller, direct transmission matrix of state-space model
$D(s)$, $\mathbf{D}(s)$	Continuous-time, linear controller
$D(z)$, $\mathbf{D}(z)$	Discrete-time, linear controller
$D_i(s)$	Continuous-time, linear controller in loop i of a multivariable system
D_{opt}, $\mathbf{D}_{opt}$	Optimal controller
$d(t)$, $\mathbf{d}(t)$	Process disturbance signal
$\hat{d}(t)$, $\hat{\mathbf{d}}(t)$	Process disturbance signal estimate
$e(t)$, $\mathbf{e}(t)$	Error signal, noise signal
$f(H)$	Average fitness of schema H in population $\boldsymbol{P}(t)$
$\bar{f}$	Average fitness of population $\boldsymbol{P}(t)$
f_i	Fitness value of string S_i

$\hat{f}_i$	Ranked fitness value of string S_i
$\tilde{f}_i$	Shared fitness value of string S_i
$f_{ij}^{(u)}(t), f_{ij}^{(l)}(t)$	Upper and lower boundary functions used in J_M
$f_u(t), f_l(t)$	Upper and lower boundary functions used in J_S
G, $\mathbf{G}$	General, linear or non-linear process (or process model)
$G(s)$, $\mathbf{G}(s)$	Continuous-time, linear process (or process model)
$G(z)$, $\mathbf{G}(z)$	Discrete-time, linear process model
$G^*(z)$	$z^k G(z)$, where k is the number of delay states of $G(z)$
$\mathbf{G}_D(s)$, $\mathbf{G}_D(z)$	Shell process model subsystem used for the disturbance estimation
$\mathbf{G}_F(z)$	Discrete-time low-pass filter used in the disturbance estimator
$G_{ij}(s)$, $G_{ij}(z)$	i, j element of transfer function matrix $\mathbf{G}(s)$, $\mathbf{G}(z)$
$\mathbf{G}_M(s)$, $\mathbf{G}_M(z)$	Shell process model subsystem used for the disturbance estimation
$\mathbf{G}_R(s)$	Shell process model subsystem used for the regulation of y_1 and y_2
$\mathbf{G}_S(z)$	Shell process model subsystem used in the Smith predictor
$\mathbf{G}_S^*(z)$	Delay-free Shell process model subsystem used in the Smith predictor
GM	Gain margin
g	Generation gap
H	Schema
$\mathbf{h}$	Size vector of hypercuboid $\mathcal{H}$
h_i	Size of the i-th edge of hypercuboid $\mathcal{H}$
$\mathbf{h}_{max}$	Size vector of the smallest hypercuboid that contains $\tilde{\mathcal{D}}$
$h_{max,i}$	The i-th element of $\mathbf{h}_{max}$
$\mathbf{I}$	Identity matrix
$I_{ij}(\mathbf{D})$	Input term of J_0
$J(\cdot)$	Single-valued objective function
$J_0(\mathbf{D})$	Proposed objective function for the Shell standard control problem
$\mathbf{J}_i(\mathbf{D})$	Objective function row vector for output i, $\mathbf{J}_i = [J_{i1} \cdots J_{iq}]$
$J_{ij}(\mathbf{D})$	Objective function element of J_M
$J_M(\mathbf{D})$	Proposed objective function for multivariable systems
$\mathbf{J}_M(\mathbf{D})$	Vector-valued version of J_M, $\mathbf{J}_M := [\mathbf{J}_1 \cdots \mathbf{J}_q]^T$
$J_S(D)$	Proposed objective function for single-input, single-output systems
K	Linear system steady-state gain
K_i	Single-input, single-output PID controller integral gain
K_{ij}	Steady-state gain of the i, j element of $\mathbf{G}(s)$
K_p	Single-input, single-output PID controller proportional gain

K_{P_i}	Proportional gain of PID controller in loop i
k	Cardinality of string alphabet, number of delay states, sample number
k_{max}	Maximum sample number, $k_{max} = t_{max}/T$
L	Linear system time delay
L_{ij}	Time delay of the i,j element of $\mathbf{G}(s)$
l	String length
l_x	Length of string element for parameter x
M	Number of strictly optimal solutions in the population
$M_H(t)$, M_H	Number of strings in population $\boldsymbol{P}(t)$ that belong to schema H
$M'_H(t+1)$	Number of strings in population $\boldsymbol{P'}(t+1)$ that belong to schema H
$M''_H(t+1)$	Number of strings in population $\boldsymbol{P''}(t+1)$ that belong to schema H
M_s	Linear closed-loop system sensitivity
N	Population size
N_i	Derivative filter coefficient of PID controller in loop i
n	Dimensions of search space
$O(\cdot)$	Computation time complexity
$O_{ij}(\mathbf{D})$	Output term of J_0
$o(H)$	Order of schema H
$\boldsymbol{P}(t)$	Population at generation t
$\boldsymbol{P'}(t+1)$	Population formed after reproduction on $\boldsymbol{P}(t)$
$\boldsymbol{P''}(t+1)$	Population formed after crossover on $\boldsymbol{P'}(t+1)$
PM	Phase margin
p	Number of inputs of multivariable process
p_c	Crossover operator probability
$p_c(H)$	Crossover survival probability of schema H
p_i	Probability of selecting string S_i for reproduction
p_m	Mutation operator probability
$p_m(H)$	Mutation survival probability of schema H
$p_r(H)$	Reproduction survival probability of schema H
q	Number of outputs of multivariable process
$r(t)$, $\mathbf{r}(t)$	Set point signal
$S_f(\cdot)$	Fitness sharing function
$S_i(t)$, S_i	i-th string in population $\boldsymbol{P}(t)$
s_i	Expected number of samples of string S_i in population $\boldsymbol{P'}(t+1)$
$\hat{s}_i$	Actual number of samples of string S_i in population $\boldsymbol{P'}(t+1)$

T	Linear system time constant, sample time
T_{D_i}	Derivative time of PID controller in loop i
T_{I_i}	Integral time of PID controller in loop i
T_{ij}	Time constant of the i,j element of $\mathbf{G}(s)$
T_{T_i}	Integral anti-windup tracking time constant of PID controller in loop i
t	Time, generation
t_{max}	Maximum simulation time
t_s	Settling time
U	Diversity measure of solutions in $\mathcal{H}$
U_0	Diversity measure of solutions in $\tilde{\mathcal{D}}$
$u(t)$, $\mathbf{u}(t)$	Process input signal
$\hat{u}(t)$, $\hat{\mathbf{u}}(t)$	Process input signal before actuator constraints
$V_{\mathcal{C}}$	Hypervolume of unit hypercube $\mathcal{C}$, $V_{\mathcal{C}}=1$
w_{ij}	Weighting coefficients used in J_M
$\mathbf{X}$	$N\times n$ matrix of parameter vectors in the population
$\mathbf{x}$	State vector of state-space model
$\mathbf{x}_i$	Parameter vector of length n that corresponds to string S_i
$\hat{\mathbf{x}}_i$	Scaled parameter vector of length n that corresponds to string S_i
x_{ij}	Parameter j of the solution that corresponds to string S_i
x_{max}	Maximum value of x
x_{min}	Minimum value of x
x_{opt}	Optimal value of x
$y(t)$, $\mathbf{y}(t)$	Process output signal
$\hat{y}(t)$, $\hat{\mathbf{y}}(t)$	Process output signal when no disturbances are present ($\mathbf{d}=\mathbf{0}$)
α	Power factor of the power law sharing function
α_{ij}	Output term weighting coefficient of J_0
β	Decay factor of the exponential sharing function
Δ_{ij}	Distance measure between strings S_i and S_j
$\Delta\mathbf{d}(t)$	Process disturbance vector change at time t, $\Delta\mathbf{d}(t)=\mathbf{d}(t_+)-\mathbf{d}(t_-)$
ΔK_{ij}	Absolute maximum uncertainty of steady-state gain K_{ij}
$\delta(H)$	Defining length of schema H
δ_i	Absolute rate limit of actuator i
ε_j	Uncertainty coefficient of steady-state gain K_{ij}
λ_{ij}	Input term weighting coefficient of J_0

η	Constraint boundary coefficient
$\bar{\sigma}(\mathbf{G})$	Largest singular value of $\mathbf{G}(s)$, with $s=j\omega$
$\underline{\sigma}(\mathbf{G})$	Smallest singular value of $\mathbf{G}(s)$, with $s=j\omega$
σ_i	Sum of the shares received by individual S_i under fitness sharing
σ_p	Population ranking selection pressure
σ_{share}	Fitness sharing radius
σ_a^2	Actuator noise variance
σ_m^2	Measurement noise variance
ω	Natural undamped frequency

1 Introduction – Survey and Thesis Outline

1.1 Introduction

This chapter begins with a brief overview of optimisation as applied to the solution of control engineering problems. Genetic algorithms (GAs) as function optimisers are then introduced, focusing on their fundamental differences and advantages over conventional algorithms. The relevance of GAs to control systems is then illustrated by a number of successful applications in different areas of process modelling and control. Finally, the project scope and the structure of this thesis are outlined.

1.2 Optimisation and Control Systems

The majority of modelling and control problems are inherently associated with function optimisation. Consider, for example, the simple closed-loop system shown in Fig. 1.1. In most cases, the controller, D, is designed such that the error signal, $e(t)$, is minimised in some desired way. The controller, therefore, acts as an optimiser which attempts to minimise some function $f[e(t)]$ on-line, by generating a suitable control sequence which is applied to the process input $u(t)$.

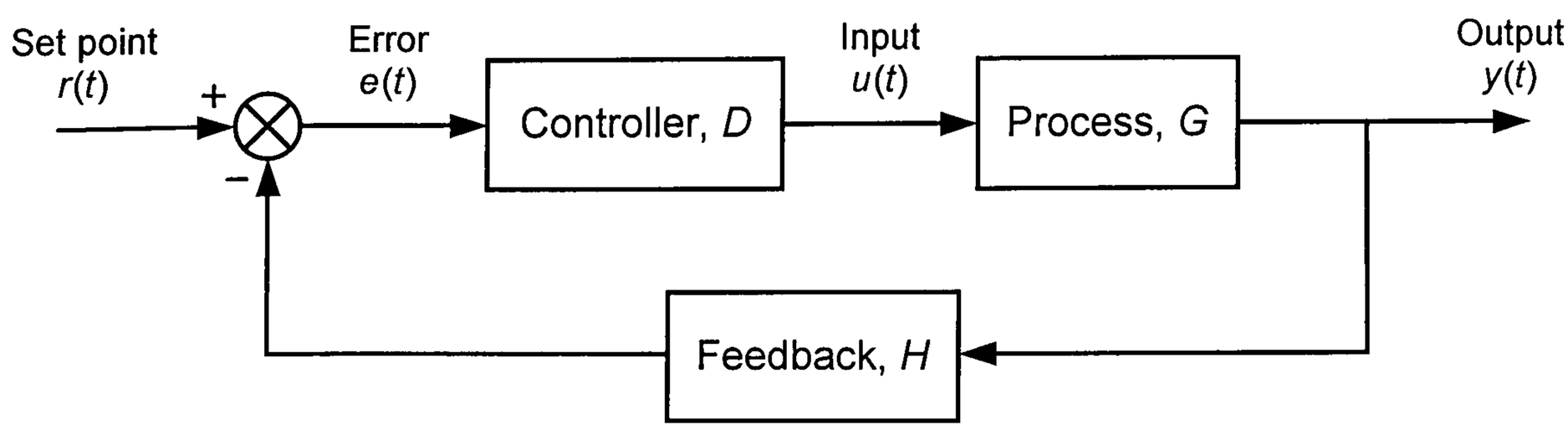


Fig. 1.1 *Simple closed-loop automatic control system*

It may often be possible to quantify the performance of the closed loop system by means of a function $J(\cdot)$, which is usually single-valued and is often called *performance index* or *objective function*. This is clearly a function of D, and the controller design problem is, therefore, equivalent to that of optimising J over the set of all permissible (usually stabilising) controllers. Depending on the complexity of the mapping defined by J and the associated control problem, the optimisation of J may be performed analytically or numerically. The numerical approach, although less elegant, is conceptually simple and has the potential to deliver excellent designs that go beyond linear, time-invariant control theory, as will become apparent later.

Several modern control approaches have been developed, which are based on function optimisation. Popular examples include the H_2 (Linear Quadratic Gaussian – LQG), and H_∞ optimal control theory, in which the H_2 and H_∞ norms are employed as performance indexes, respectively. A comprehensive treatment of H_2 and H_∞ optimal control theory can be found in Zhou, Doyle, and Glover (1996). Other popular optimisation approaches to control include Model Reference Adaptive Control (MRAC), in which the controller parameters are optimised on-line, so that the process output asymptotically matches that of a reference model. For more information on MRAC, the reader is referred to Åström and Wittenmark (1995). Most predictive control algorithms (García and Morshedi, 1986; Clarke, Mohtadi, and Tuffs, 1987a, 1987b; García, Prett, and Morari, 1989; Clarke and Mohtadi, 1989) are also based on on-line function optimisation. Worth mentioning is the so-called Edmunds' algorithm (Edmunds, 1979), in which the controller parameters are optimised to make the closed-loop transfer function approach some target transfer function as closely as possible over a specified frequency range.

Optimisation has also been employed for the tuning of PID controllers. Lopez, Miller, Smith, and Murrill (1967) used a number of performance indexes based on integrals of functions of the form $f[t, e(t)]$, such as the Integrated Squared Error (ISE) criterion, to develop graphs that relate the optimal P, PI, and PID settings with the three parameters of a first-order dead time process model. A time-domain PID tuning method that is also based on integral performance criteria was proposed by Dan-Isa and Atherton (1997). More recently, Gagnon, Pomerleau and Desbiens (1999) proposed a decentralised PI tuning method for multivariable processes, based on the minimisation of an objective

function which is derived from standard μ-synthesis (structured singular value) theory. This approach results in PI controllers that achieve improved robust performance in the face of process uncertainty and controller output variation constraints.

1.2.1 Advantages of Optimisation-Based Controller Design

In conventional, linear controller design methods, the problem specifications are usually expressed in terms of standard system characteristics, such as gain and phase margins, locations of poles and zeros, bandwidth, peak overshoot, settling time, rise time, and others. Although such characteristics may be sufficient in ensuring that the closed-loop system is stable and well-behaved, they may not accurately represent the specifications that can arise in an arbitrary, real-world control problem. Furthermore, in cases where the closed-loop system contains non-linear elements, linear system characteristics such as those mentioned above can often become meaningless. In a function optimisation framework, however, linearity is not at all a prerequisite. A performance index $J(\cdot)$ that accurately reflects the given specifications can be formulated, and then optimised using a suitable analytical or numerical method.

Analytical optimisation is usually a very difficult task, mainly due to the complexity of the mapping defined by J and the associated control problem. However, the widespread availability of high-speed computers has made numerical optimisation a viable (albeit less elegant) alternative to analytical optimisation. The numerical optimisation approach is conceptually simple in that it only requires the numerical solutions of the differential and difference equations associated with the closed-loop system. These can be obtained using most standard control system simulation packages, provided a suitable model of the controlled process exists. The process model may be derived from first principles or from input/output data, and can contain both linear and non-linear elements. Objective function J for a candidate controller D can then be numerically evaluated by simulating the closed-loop system. A suitable optimisation algorithm may be used to optimise J, in order to obtain the optimal controller, D_{opt}.

Clearly, optimisation-based controller design has the important advantage that it can be applied to a wide range of complex, non-linear control problems which cannot normally be solved reliably using conventional design methods. Another advantage is the ability

to handle arbitrary performance specifications. The fact that numerical optimisation does not rely on analytical tools makes this approach extremely flexible. For example, a new (linear or non-linear) element in the closed-loop system can be introduced by simply including it in the system simulation code. Similarly, a new design specification can be introduced by simply modifying J accordingly. The numerical optimisation can then be restarted with practically no further code modifications.

1.2.2 Limitations of Optimisation-Based Controller Design

The major limitation of optimisation-based controller design methods is the fact that they all depend on the performance index $J(\cdot)$ and the way in which it quantifies the problem specifications. Care must, therefore, be exercised when formulating J, because a poorly designed performance index can result in controllers which are mathematically optimal, but are unacceptable in practice. The performance indexes employed in most analytical optimisation methods, such as H_2 and H_∞ optimal control theory, are usually continuous functions whose derivatives exist and can be expressed analytically in relatively simple terms. The problem specifications, however, may often be too complex to be adequately described by such simple objective functions. This results in designs that may require several *ad hoc* controller adjustments before they can be acceptable in practice.

Another limitation is that, although J may adequately reflect the problem specifications, it may be too complex to be optimised, even in a numerical framework. In cases where J contains many local optima and is discontinuous, most conventional, calculus-based numerical optimisation algorithms are expected to converge to sub-optimal solutions, or not converge at all in some cases. Another difficulty is that the computations required to evaluate J may easily become excessive, and numerical robustness problems may also arise. The choice of a suitable optimisation method becomes a very important issue in such cases.

1.3 Genetic Algorithms

Genetic algorithms (GAs) are stochastic global search methods that are loosely based on the metaphor of natural biological evolution. They maintain a set of candidate solutions to a given problem, which are left to 'evolve' using artificial genetic operators such as

reproduction, *crossover* and *mutation*. GAs work by combining the Darwinian 'survival of the fittest' principle with a probabilistic information exchange strategy inspired by the processes of natural genetics, to form a *structured yet randomised* search algorithm that promises to be highly capable of identifying optimal or near-optimal solutions to a wide range of search, optimisation and machine learning problems. GAs have been developed by John Holland, his colleagues, and his students at the University of Michigan. Studies by Holland (1975), De Jong (1975), Goldberg (1989a), and others have demonstrated, both theoretically and experimentally, the superior performance of GAs over traditional search methods. They are currently being employed in a wide range of domains. More information on GAs and a list of practical applications can be found in Holland (1992), Fogel (1994), Goldberg (1994), and the introductory textbooks by Goldberg (1989a), and Mitchell (1996). Because of their unique structure and operation, GAs differ from more traditional search procedures in some very fundamental ways, making them ideal candidates as global function optimisers. The most important differences between GAs and conventional search methods are summarised in the following paragraphs. A more thorough comparison between GAs and conventional, gradient-based methods can be found in Salomon (1998) and the references therein.

Most conventional optimisation methods are local in scope, mainly because they search from a single starting point. They can, thus, converge to a sub-optimal solution in the neighbourhood of the starting point, and miss the global optimum which may be located elsewhere. GAs, however, search from a *population of points*, not a single point, thus dramatically increasing the probability of reaching the global optimum. This makes them suitable for the optimisation of multimodal functions (i.e. functions which contain many local optima).

Another weakness of conventional optimisation methods is that they usually rely on the existence of the partial derivatives of the objective function. Derivative information is used to guide the search towards the optimal point in some local neighbourhood. Popular examples of this approach include the so-called 'hill-climbing' methods. GAs, however, *only need objective function information*. This enables them to be applicable in domains where continuity and derivative existence do not apply. They can also be used for the optimisation of 'noisy' functions, where the same set of parameters will not, in general, produce exactly the same result.

Another characteristic of GAs that distinguishes them from most conventional search methods is that they work with a *coding of the parameter set* and not with the parameters themselves. This makes them directly applicable to a very wide range of non-numerical, discrete, combinatorial, and mixed optimisation problems. Most conventional optimisers based on continuous parameter variations cannot normally be used for the solution of such problems.

Finally, unlike many conventional search methods, GAs use *probabilistic transition rules* to guide their search, not deterministic ones. This may seem odd to those familiar with deterministic methods, but the use of probabilistic processes does not suggest that GAs are equivalent to some simple random search. It can be shown (Goldberg, 1989a) that a clear distinction exists between the stochastic operators of GAs and other methods based on simple random walk. Using chance to achieve highly directed results may seem unusual, but the same mechanism is also used by nature, with obvious success.

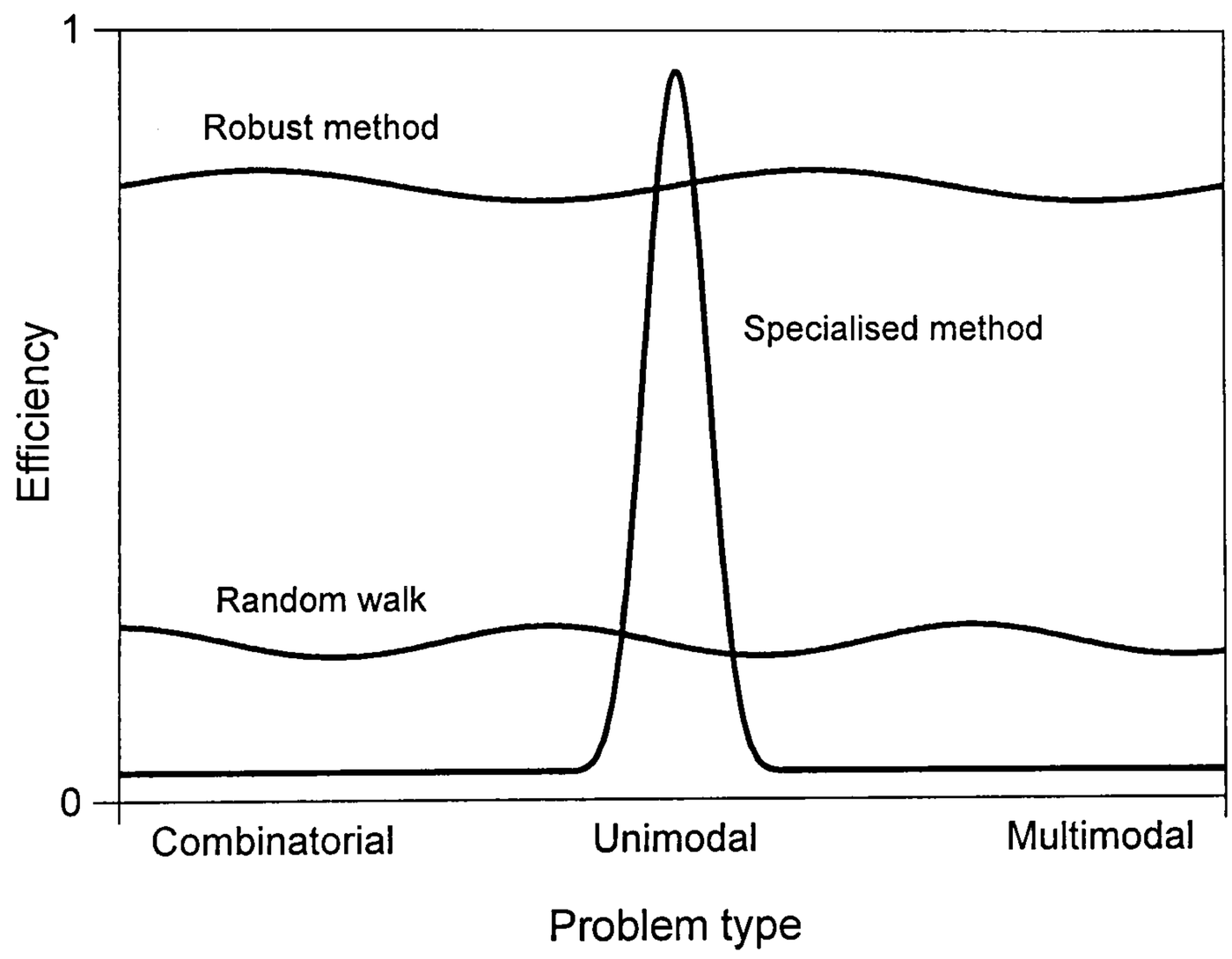


Fig. 1.2 *Efficiency of different classes of search methods across a problem continuum (Goldberg, 1989a)*

The power of GAs comes from the fact that they are *robust* and thus have the potential to be applicable to a wide range of problems, including those which other methods find difficult to solve efficiently. As expected, GAs are not guaranteed to find the globally optimal solution to any given problem, but they are generally good in finding "acceptably

good" solutions to a wide range of problems "acceptably quickly". The importance of robustness in a search method can be put in better perspective by observing Fig. 1.2. It can be seen that a specialised method performs very well in the problem area it has been designed for, but its efficiency drops rapidly when applied in different problem areas. In contrast to that, purely randomised methods such as random walk perform consistently in a wide range of problem areas, but their efficiency is generally low. Robust methods such as GAs combine efficiency with consistency, achieving acceptable performance across a wide range of domains. Therefore, in cases where specialised methods exist for solving specific problems, these methods are likely to outperform GAs, but in difficult areas where no specialised methods exist, GAs can provide a very effective — if not the only — approach. Even in cases where methods exist and work well, improvements can be made by hybridising them with GAs (Goldberg, 1989a; Salomon, 1998).

1.4 Genetic Algorithms and Control Systems

It is evident that, as a robust means for optimisation, the genetic algorithm approach fits well within the scope of optimisation-based process modelling and control, where noisy, highly non-linear, multimodal, and discontinuous functions of many dimensions need to be optimised. An overview of the relevance of GAs to problems in control engineering can be found in Chipperfield and Fleming (1995), Zalzala and Fleming (1996), Linkens and Nyongesa (1996b), and the references therein. GAs have already been employed for the solution of modelling and control problems, with a high degree of success. A number of successful applications of GAs to control systems are presented in this section.

1.4.1 Genetic Algorithms in Process Modelling

GAs have been employed for the parameter estimation and structure selection of both linear and non-linear system models. Kristinsson and Dumont (1992) demonstrated the use of GAs for the parameter estimation of both continuous-time and discrete-time linear systems, and for identifying the poles and zeros or the physical parameters of a system. Tan and Li (1997) employed GAs for the identification of linear and non-linear models from process step response data. Fonseca and Fleming (1996) used a multiobjective GA to identify non-linear polynomial models for a real non-linear system. A similar method was proposed by Li and Jeon (1993) who then used it in a learning control scheme (Li,

Tzeng, and Jeon, 1997). Other genetic approaches to non-linear modelling which also employ polynomial models were proposed by Rodriguez-Vazquez and Fleming (1997, 1998) who identified optimal non-linear model structures using the technique of Genetic Programming (GP), developed by Koza (1992). GP was also used by Gray *et. al.* (1998), and McKay, Willis, and Barton (1997), for the identification of general non-linear model structures. The use of GP enables complete mathematical models to be constructed by optimally combining different types of linear and non-linear principal components. The obtained models have the advantage that they more closely resemble those obtained by first principles, and can thus provide more structural insight into the modelled process characteristics than black-box models such as neural networks. French, Cox, and Ho (1997) proposed a GA-based framework for the identification of the structure, order, and parameters of multivariable discrete-time transfer function matrices. GAs have also been used for the identification of fuzzy models. Recent applications in this direction include Rahmoun and Benmohamed (1998), and Hwang (1999). Shieh, Zheng, and Wang (1997) utilised GAs to identify discrete-time parametric models for uncertain processes which provide less conservative results than those obtained by conventional methods. Alonge, D'Ippolito, Ferrante, and Raimondi (1998) employed GAs to determine the mechanical and electrical parameters of a real induction motor model, and Liu and Kadirkamanathan (1999) proposed a GA-based method that addresses the problems of structure selection and identification of neural network models for non-linear systems in a multiobjective optimisation framework. Finally, Hong and Billings (1999) proposed a method for the parameter estimation of linear-in-the-parameters non-linear models, based on stacked regression and an evolutionary algorithm.

1.4.2 Genetic Algorithms in Process Control

Several GA-based methods have been developed for the optimal design and tuning of controllers. Porter and Jones (1992) used GAs to tune discrete-time Proportional plus Integral plus Derivative (PID) controllers for multivariable processes. Wang and Kwok (1994) proposed a GA-based technique for the tuning of PID controllers for non-linear processes, using the ISE performance criterion. Kawabe, Tagami, and Katayama (1996), and Kawabe, Tagami, and Okamura (1998) developed GA-based methods for the design of PID controllers that achieve robust stability and performance in the presence of plant uncertainty. An adaptive GA-based learning algorithm for the automatic tuning of PID

controllers as applied to heating, ventilating, and air conditioning systems, was proposed by Huang and Lam (1997), and Warwick and Kang (1998) proposed a self-tuning PID control scheme which they applied to the on-line control of a real industrial plant. The proposed approach is based on a hybrid scheme which combines the Recursive Least Squares (RLS) algorithm with a GA. Zuo (1995, 1997) proposed a GA-based method for the design of discrete-time set point tracking adaptive PID controllers for complex multivariable plants. Finally, GA-based methods for the design of robust PID controllers using H_∞ optimal control theory can be found in Takahashi, Peres, and Ferreira (1997), and Chen and Cheng (1998).

GAs have also been applied in more complex control strategies. Varšek, Urbančič, and Filipič (1993) employed GAs to derive and fine-tune a set of *if-then* rules for the control of dynamic systems, without prior knowledge about the system to be controlled. Filipič, Urbančič, and Križman (1999) proposed a similar rule-based approach. Krishnakumar and Goldberg (1992) used GAs to solve aerospace-related control problems by means of optimising standard linear quadratic regulators. Kristinsson and Dumont (1992) applied GAs to the design of discrete-time pole placement adaptive controllers. A comparative analysis of the performance of various GA-based adaptive control methods relative to conventional methods can be found in Lennon and Passino (1999). Hunt (1992a, 1992b) proposed an approach for the synthesis of LQG and H_∞ optimal controllers for linear systems based on GAs. Patton and Liu (1994), and Clarke and Davies (1997), employed GAs for the robust control design of multivariable systems based on the eigenstructure assignment methodology. Shieh, Wang, and Tsai (1999) proposed a state-space design methodology for the optimal design of discrete-time parametric uncertain systems, based on GAs. Another application of GAs to robust control system design can be found in Marrison and Stengel (1997). GAs have also been employed as optimisers in predictive control algorithms. Onnen *et. al.* (1997) applied GAs to model-based predictive control of a non-linear system with input saturation and rate limit constraints. Martínez, Senent, and Blasco (1998) presented a Generalised Predictive Control (GPC) scheme that uses GAs to optimise the associated objective function. A practical application of GAs to the optimisation of the control parameters of a pneumatic servo cylinder drive is reported in Jeon, Lee, and Hong (1998). French, Cox, and Ho (1997) proposed a GA-based solution to the input/output pairing problem in multivariable processes. Patton, Chen, and Liu (1997) developed a new approach to the design of robust fault detection systems based

on GAs. In the area of process optimisation, a GA-based method was proposed by Pham (1998) for the constrained optimisation of chemical engineering processes, and Santos and Dourado (1999) developed a GA-based method for the global optimisation of energy and production in process industries. GAs have also been employed in the area of fuzzy logic control. Refer to Linkens and Nyongesa (1995a, 1995b, 1996a), Trebi-Ollennu and White (1997), Pham and Karaboga (1997), Lian, Marzuki, and Rubiyah (1998), Herrera, Lozano, and Verdegay (1998), Gürocak (1999), Wong and Fan (1999), and Hwang (1999) for a few examples.

Most approaches to optimisation-based controller design and tuning use single-valued objective functions to provide the necessary performance indexes to guide the search for optimal solutions. However, control engineering problems are very seldom associated with a single objective. Instead, several, often conflicting objectives are usually present, thus resulting in vector-valued objective functions. Such cases are usually treated by weighting and combining all objectives into a single-valued function, thus transforming them into single-objective optimisation problems. This approach may be acceptable in certain cases, but there are times when combining the objectives in an efficient way may not be practically feasible. Although GAs are inherently unsuitable for multiobjective optimisation in their standard form, a number of extensions have been proposed which enable them to efficiently optimise vector-valued objective functions in a multiobjective framework. An overview of a number of such extensions can be found in Fonseca and Fleming (1995). Fonseca and Fleming (1993, 1998a, 1998b) proposed what is known as the Multiobjective Genetic Algorithm (MOGA). The relevance of multiobjective GAs to control systems is outlined in Fonseca and Fleming (1994). The MOGA approach has successfully been employed in a number of control problems. Chipperfield and Fleming (1996) used a MOGA to design a multivariable control system for a gas turbine engine, where both the structure and the parameters of the controller are optimised. Another application is reported in Thompson, Chipperfield, Fleming, and Legge (1999).

1.5 Project Scope

The main objective of this research project is to develop general methods for the optimal design of controllers for multivariable processes. The project focuses on design methods based on numerical optimisation, where GAs are employed to optimise a number of

specially formulated objective functions. Strong emphasis is given on the generality and open architecture of the proposed methods, which must be applicable to a wide range of real-world multivariable control problems involving non-linearities, noise, and arbitrary performance specifications and controller structures. Other important aims of this project are to extend standard GAs, in order to improve their efficiency in problems associated with optimisation-based control, and also to justify the choice of GAs in the proposed optimisation framework by statistically comparing them with a number of conventional function optimisation algorithms.

1.6 Thesis Outline

The structure of this thesis is outlined below. Most of the material contained in Chapter 2 is standard and is only intended as a brief review of the current state of affairs in the field of GAs as function optimisers. The main contributions and novel aspects of this work are contained in Chapters 3 to 6 and are summarised in Chapter 7.

Chapter 2 – Genetic Algorithms and Function Optimisation

This chapter begins with a brief introduction to function optimisation, which is central to the design methods developed in this work. GAs are then introduced, and their main features and advantages over conventional optimisation methods are outlined. GAs are then treated in a more systematic and rigorous fashion, in order to establish key results that provide a deeper insight into their operation, and enable a thorough and quantitative assessment of their performance.

Chapter 3 – Analysis and Design of Objective Functions for Control Systems

This chapter is primarily concerned with the analysis and design of objective functions, as applied to the solution of control engineering problems. A novel objective function is proposed for single-input, single-output (SISO) processes, that overcomes many of the weaknesses of conventional objective functions. The proposed objective function is then experimentally analysed using a simple two-term parametric controller tuning problem. The obtained results are analysed and compared with those obtained using conventional objective functions, as well as using a number of conventional tuning methods. Finally,

the proposed objective function is generalised to treat multi-input, multi-output (MIMO) control problems.

Chapter 4 – Locating Multiple Optimal Solutions Using Genetic Algorithms

The objective functions developed in Chapter 3 have the important characteristic that they result in a *family* of solutions that completely satisfy the problem specifications. This results in a (usually infinite) set of optimal solutions. Standard GAs are known to have problems in identifying multiple optimal solutions, because the population usually converges to a small subset of the entire optimal solution set. In this chapter, GAs are extended using a novel fitness assignment strategy called *adaptive fitness sharing*. It is shown that the proposed method enables the GA to evolve a population whose members are *almost uniformly distributed* within the optimal solution set. The proposed method is not limited to the optimisation of the objective functions developed in this work, and can be used in many different search landscapes containing multiple optimal solutions.

Chapter 5 – Decentralised PI Controller Tuning for Multivariable Processes

In this chapter, a new method for the automatic tuning of decentralised PI controllers for multivariable processes is proposed, based on GAs. The main advantage of the proposed method is that it gives the designer the freedom to *explicitly* specify the performance specifications associated with a given control problem, in terms of time-domain bounds on the closed-loop responses. This is achieved by transforming the PI controller tuning problem into a function optimisation problem by means of the MIMO objective function developed in Chapter 3. GAs are then employed for the minimisation of this function, and the method of adaptive fitness sharing developed in Chapter 4 is utilised, in order to maximise the diversity of the obtained family of optimal solutions. Simulation results are presented to illustrate the effectiveness of the proposed method. The choice of GAs as a suitable optimisation method is experimentally supported by statistically comparing them with two conventional optimisation methods.

Chapter 6 – Solution to the Shell Standard Control Problem

In this chapter, a new solution to the Shell standard control problem is presented, based on GAs. The proposed control scheme includes two discrete-time PID controllers with

integral anti-windup and a multivariable Smith predictor to provide the required process output regulation, while the process input minimisation problem is analytically solved on-line, by estimating the unmeasured disturbances entering the process and solving the associated linear program. GAs are successfully applied to the automatic tuning of the PID controllers according to the given specifications, using an extension of the MIMO objective function developed in Chapter 3. Extensive simulation results are presented to demonstrate the effectiveness of the proposed control scheme.

Chapter 7 – Conclusions – Main Contributions and Further Work

The key outcomes and main contributions of this research project are summarised in this chapter, and a number of suggestions for further work, that will extend the application of GAs in the area of control systems engineering, are given.

1.7 Summary

In this chapter, an overview of function optimisation as applied to the solution of control engineering problems was given, followed by a brief introduction to genetic algorithms (GAs) as global function optimisers, with emphasis on their fundamental differences and advantages over conventional search algorithms. A literature survey was then presented, indicating the relevance of GAs to process modelling and control problems. Finally, the project scope and the structure of this thesis were outlined.

2 Genetic Algorithms and Function Optimisation

2.1 Introduction

This chapter begins with a brief introduction to function optimisation, which is central to the design methods developed in this work. Genetic algorithms as function optimisers are introduced, and their main features and advantages over conventional function optimisation methods are outlined. Genetic algorithms are then treated in a more systematic and rigorous fashion, in order to establish key results that provide a deeper insight into their operation, and enable a quantitative assessment of their performance.

2.2 Typical Search Spaces in Function Optimisation

A function can be thought of as a mapping from a set of elements to another set of elements. For example, the function $f(x) = x^2$, $x \in \mathbb{R}$ maps all real numbers to all real non-negative numbers (its domain and codomain are sets $\mathbb{R}$ and $\mathbb{R}_+$ respectively). In a typical function optimisation problem, the requirement is to find points in the domain of the function associated with the problem, that satisfy certain optimality criteria. In most cases, the function to be optimised, often called *objective function* or *cost function*, is constructed in such a way that the desirable points in its domain are the ones at which the function attains its global extremum (maximum or minimum value). In the context of function optimisation, a function and its domain and codomain form a *search space*. Finding the global extremum in a search space is, in general, a very difficult problem. Therefore, it is crucial that the algorithm used for the optimisation is able to explore the search space, and at the same time exploit certain properties of the search space, which can help direct the search towards the global extremum. Some typical search spaces that can occur in a function optimisation problem are shown in Fig. 2.1.

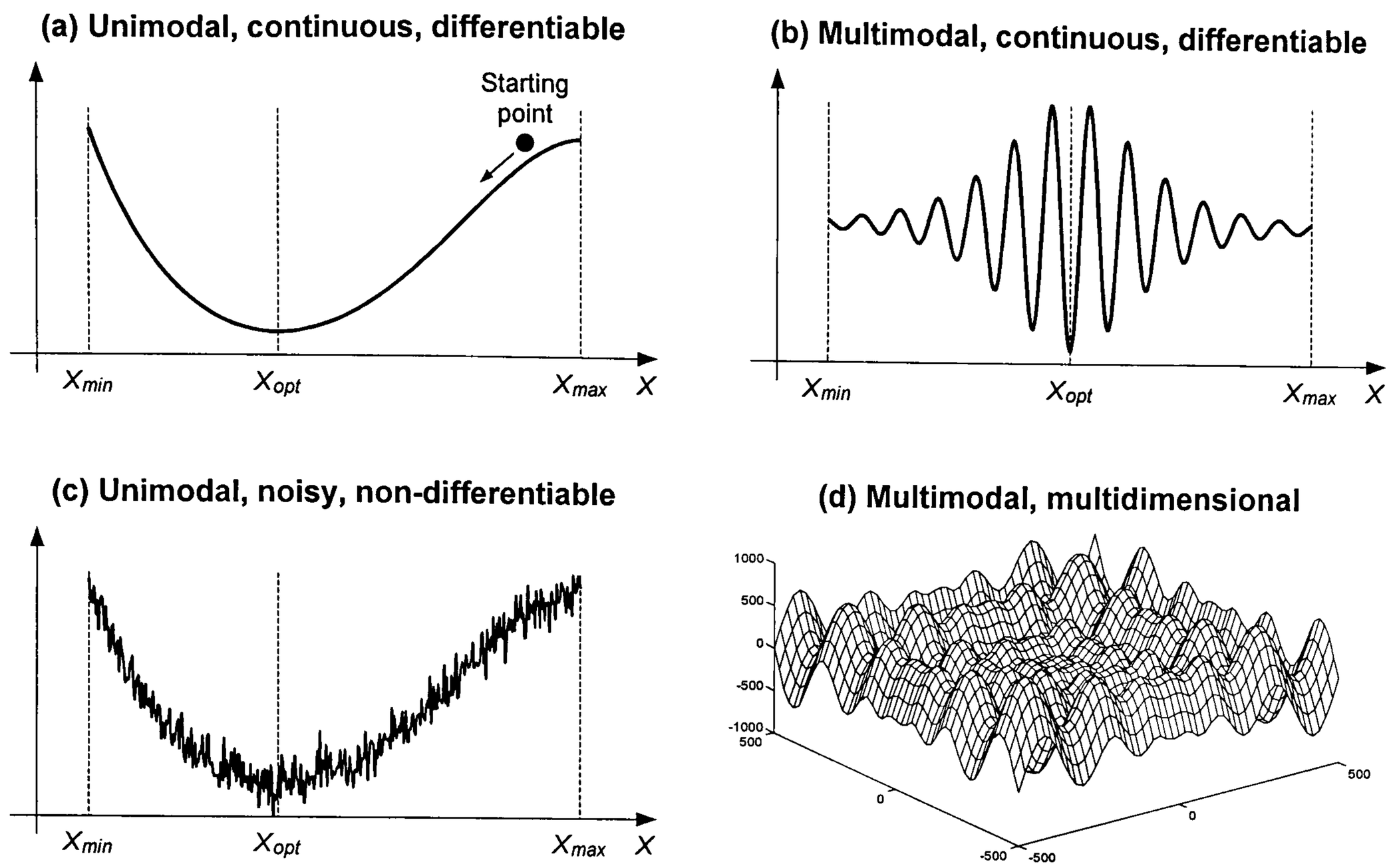


Fig. 2.1 Typical search spaces in function optimisation

Search space (a) in Fig. 2.1 results from a smooth (continuous, differentiable) and unimodal function, the minimisation of which is a trivial task. Since the derivatives exist at all points in the search space, they can be used to guide the search, so that the optimal solution, x_{opt}, is found within very good accuracy. Search space (b) results from a smooth (continuous, differentiable), but multimodal function. Derivative information can still be used to guide the search, but locating x_{opt} is now more difficult because many sub-optimal solutions (local minima) exist. It is, therefore, crucial that the starting point is chosen in the neighbourhood of x_{opt}. Although qualitatively unimodal, search space (c) is difficult to optimise due to the presence of noise which makes the associated function non-differentiable. Formally, 'noisy' search space (c) does not correspond to a function since each point x in $[x_{min}, x_{max}]$ can take an infinite number of values (the same set of parameters will not, in general, produce exactly the same result). Finally, search space (d) results from a two-dimensional, multimodal function.

Real-world search spaces are often of high dimensionality, multimodal and corrupted with noise, properties that can render most calculus-based and other conventional optimisation methods inapplicable.

2.3 Conventional Optimisation Algorithms

Most conventional function optimisation algorithms can be grouped into the following main categories. More details on the optimisation methods discussed in this section can be found in Rao (1996).

- Direct and indirect gradient-based algorithms
- Region-elimination methods
- Polynomial approximation methods
- The Nelder-Mead downhill simplex method
- Randomised algorithms

Direct and indirect gradient-based algorithms

An obvious prerequisite of gradient-based algorithms is for the function to be differentiable and hence continuous. Direct gradient-based methods restrict the search space to the points where the partial derivatives in all directions are zero. This is only possible when an analytical expression for the function is available, something extremely rare in real-world problems. The indirect gradient-based methods use the numerical partial derivatives of the function at a given starting point, in order to guide the search towards other points in the neighbourhood of the starting point where all partial derivatives are zero. These are the so-called *hill-climbing* methods. The applicability of such methods is limited because of their dependence on the existence of derivatives. Another major disadvantage is that they are local optimisers since they can easily converge to local optima in the case of multimodal functions.

Region-elimination methods

Region-elimination methods are only applicable to unidimensional search spaces. They locate the optimal point contained in a given search interval $[x_{min}, x_{max}]$ in the domain of a function $f(x)$, by successively eliminating sub-optimal regions in this interval, thus narrowing the interval bracketing the optimal point. When the bracketing interval is sufficiently small, the search terminates. Commonly used region-elimination methods include *interval halving, dichotomous search, golden section search, Fibonacci search,*

and others. These methods are local in scope, and are only applicable to the optimisation of locally unimodal functions of only one variable. Their application is, therefore, limited to small-scale optimisation problems.

Polynomial approximation methods

This is another class of methods used for the optimisation of unidimensional functions. They locate a point in the neighbourhood of the optimal point by extrapolation and interpolation using polynomial approximates as models of $f(x)$. The candidate optimal points are then derived analytically using the resulting polynomial. Both quadratic and cubic approximations have been proposed. There is evidence that these methods perform slightly better in practice than the region-elimination methods described earlier. Polynomial approximation methods are local optimisers that are applicable to the optimisation of sufficiently smooth functions of only one variable, properties which limit their scope of application.

The Nelder-Mead downhill simplex method

The *downhill simplex method* is due to Nelder and Mead (1965). This is an entirely self-contained and relatively simple multidimensional search method, which does not require the existence of derivatives. Convergence to a (local) optimum is guaranteed, and is achieved by appropriately modifying a n-dimensional simplex in an iterative fashion. The *simplex method* of linear programming also makes use of the geometrical concept of a simplex, but is otherwise unrelated to the downhill simplex method described here. The downhill simplex method is very easy to implement, only requires function evaluations, and can be used in multidimensional search spaces, but suffers from a slow convergence and can also easily converge to local optima.

Randomised algorithms

Randomised algorithms such as *random walk* have the advantage that they can be used in almost all types of search spaces, since they make almost no assumptions about $f(x)$. However, they are extremely inefficient because they do not exploit the search space, and an extremely large number of function evaluations is usually required in order for them to converge to the optimal region within reasonable accuracy.

It is clear that there is a trade-off among the different algorithms, between the degree of *exploration* of the search space and the degree of *exploitation* of the available information about the search space. The gradient-based algorithms have a high degree of exploitation by using the derivatives of the function (if they exist) to guide the search, but can easily converge to local optima because of their lack of exploration of the search space. On the other hand, randomised algorithms have a high degree of exploration of the search space (it is theoretically guaranteed that, given a sufficiently large number of function evaluations, a near-optimal solution will be found), but they do not sufficiently exploit the available search information. This means that, even if they reach a point in the neighbourhood of the optimal solution, they may easily diverge to other sub-optimal points in the search space. An in-depth treatment of conventional optimisation methods can be found in Rao (1996), Fletcher (1987), and Gill, Murray, and Wright (1981).

2.4 Overview of Genetic Algorithms

Genetic algorithms (GAs) are global, stochastic search methods that are based on natural population genetics. They maintain a set of many candidate solutions to a given problem, which 'evolve' using genetic operators such as *reproduction*, *crossover* and *mutation*. Studies by Holland (1975), De Jong (1975), Goldberg (1989a), and others have demonstrated, both theoretically and experimentally, the superior performance of GAs over traditional optimisation techniques. Due to their generality and robustness they are now being applied to a wide range of domains. The material presented in this section is standard and well-known. For more details and a list of practical applications, the reader is referred to Holland (1992), Fogel (1994), Goldberg (1994), and the introductory textbooks by Goldberg (1989a), and Mitchell (1996).

Simple GAs operate on a set of N candidate solutions. This set is often called the *population*. Each solution is simply a set of parameters associated with the optimisation problem. Each parameter is encoded as a string element of cardinality (number of alphabet characters) k. All string elements associated with a solution are concatenated to form one long *string* of length l (which is directly analogous to a *chromosome* in natural genetics). In most cases, and for reasons that will become apparent later in this chapter, binary strings ($k=2$) are used in the encoding. An example of a typical binary string that can be used in a simple GA is shown in Fig. 2.2 below.

Parameter x Length l_x	Parameter y Length l_y	Parameter z Length l_z
0110100100111100 ... 11	0000101100111101 ... 01	1011111010110100 ... 10

Fig. 2.2 *Example of a string representing a solution to the optimisation of function $f(x,y,z)$*

In the above example, the function to be optimised is $f(x,y,z)$. The parameters are firstly converted to binary string elements of lengths l_x, l_y, and l_z. The longer the length of each string element, the higher the resolution of the corresponding parameter. All string elements are then concatenated to form one long string of length $l = l_x + l_y + l_z$.

In the beginning of the evolution process, the strings in the population are usually chosen at random. If there is prior knowledge about the locations of near-optimal regions in the search space, it may be helpful to initialise part of the population with strings in these regions. The initial population then evolves in generations, using a sequence of genetic operators. In a simple GA, three fundamental genetic operators are used in the evolution process. These are *reproduction*, *crossover* and *mutation*.

2.4.1 Reproduction

Each one of the N strings in the population is firstly decoded to its corresponding set of parameters. These parameters are then used to evaluate the performance of each string by means of an *objective function*, which is a problem-dependent, real and single-valued function of the parameters to be optimised. For each string, the performance index obtained from the objective function is used to produce a non-negative real number, whose value is called the *fitness value* of the string and is a measure of the quality of the corresponding solution. During the reproduction phase, individual strings are selected according to their fitness values. This means that strings with higher fitness values have a higher probability of producing one or more offspring in the next generation. Fitness values are often chosen to be directly proportional to the objective function values. This scheme is called proportionate selection. Other, more advanced selection schemes exist, including rank selection, tournament selection, truncation selection, and others. More details on proportionate and rank selection can be found in Section 4.4 of Chapter 4. Convergence models for a number of selection schemes can be found in the work by Thierens and Goldberg (1994). The reader is also referred to Goldberg and Deb (1991), and Hancock (1994), for comparisons between different schemes. After reproduction, the selected strings are placed in a mating pool where crossover takes place.

2.4.2 Crossover

The crossover operator is of major importance in the operation of a GA as it enables the *exchange of information* between strings. Initially, pairs of the reproduced strings in the mating pool are formed at random. In single-point crossover, a crossing site is randomly chosen along the length of each string pair. Then, a new pair of strings is created by swapping all characters (usually called *features* or *detectors*) to the left or right of the crossing site. This is illustrated in Fig. 2.3 below.

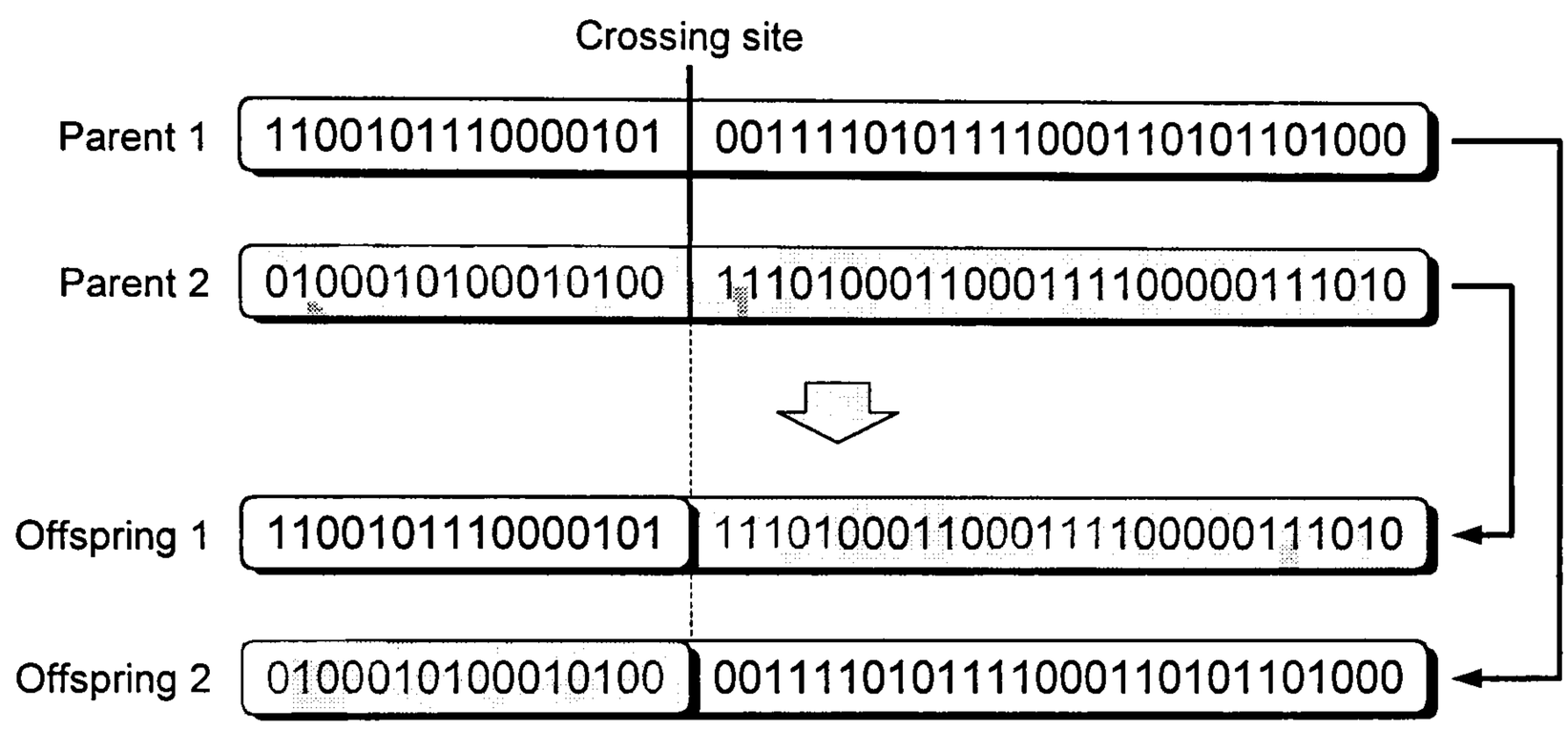


Fig. 2.3 *Example of single-point crossover*

In this way, good qualities of highly fit strings can be combined to form new, possibly better strings. The crossover operator is applied with a given probability called the *crossover probability*, p_c. Therefore, on average, Np_c strings undergo crossover. It should be noted that more advanced crossover operators have been proposed, including multipoint crossover (De Jong, 1975), uniform crossover (Syswerda, 1989), shuffle crossover (Caruana, Eshelman, and Schaffer, 1989), and others.

2.4.3 Mutation

Mutation is the random alteration of the value of a string position and usually occurs with a small probability called the *mutation probability*, p_m. Consequently, on average, Nlp_m mutations occur per generation. Mutation plays a secondary but important role in a GA, and is needed to replace potentially useful genetic material that has been lost and cannot be replaced by crossover alone. It ensures the reachability of all points in the search space. The mutation operator is illustrated in Fig. 2.4 below.

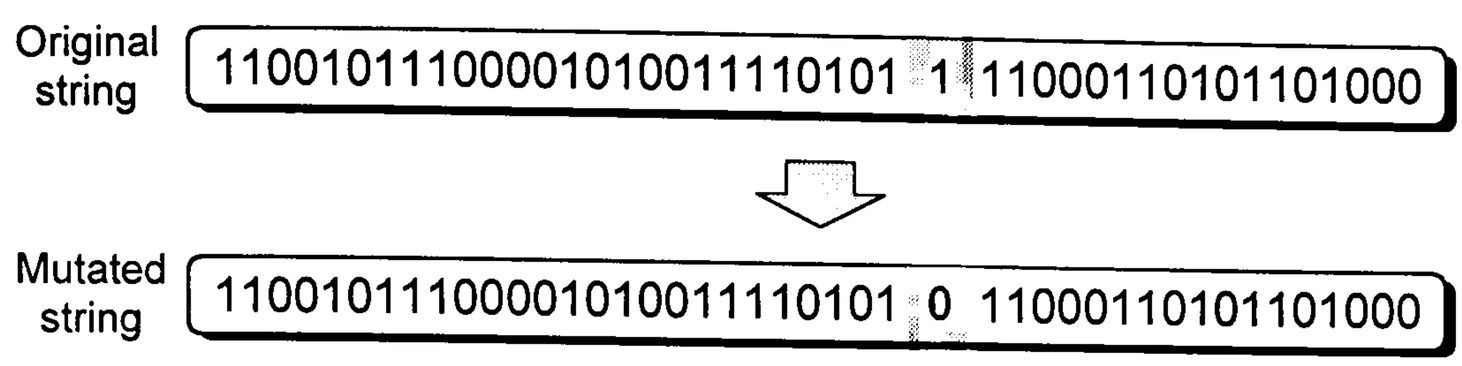


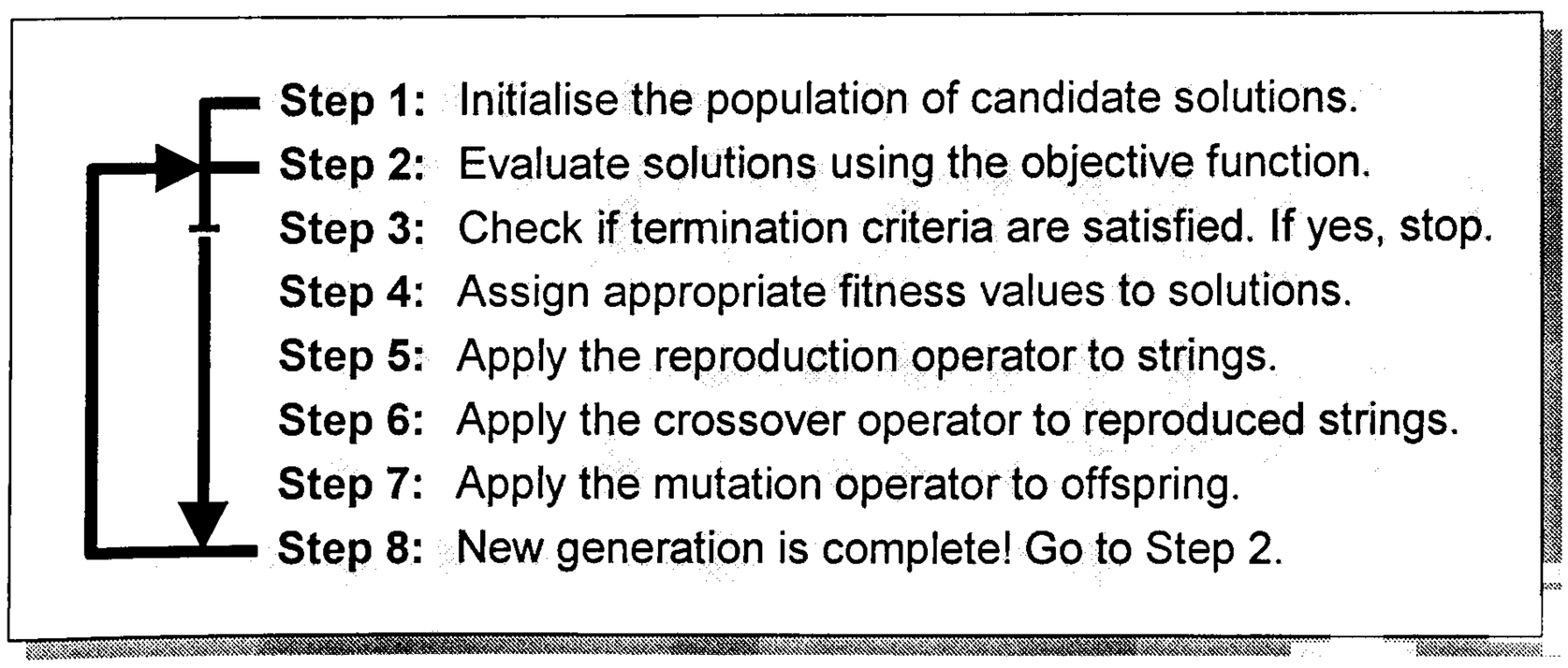
Fig. 2.4 *Example of uniform mutation*

2.4.4 Generation Gap and Elitism

In a simple GA, strings in the population undergo reproduction, crossover, and mutation, and the entire population is replaced in each generation. Rudolph (1994) has shown that this procedure cannot ensure asymptotic convergence to an optimum. This problem can be overcome by employing a heuristic technique first proposed by De Jong (1975), in which only a percentage of the population is replaced in each generation. This is controlled by the so-called *generation gap*, g, which can take values between 0 and 1. That is, $N(1-g)$ strings in the population are chosen to survive intact in the next generation. These strings can be chosen at random (random reinsertion) or according to their fitness values (fitness-based reinsertion). The latter is often called an *elitist strategy* because the $N(1-g)$ *best* individuals in the population always propagate through to successive generations. This guarantees asymptotic convergence by ensuring that the best solution in a generation can only be replaced by a better solution.

2.4.5 Structure of the Simple Genetic Algorithm

The algorithmic structure of the so-called *Simple Genetic Algorithm (SGA)* can be summarised in the following sequence of steps.



It can be seen that genetic algorithms are very different from conventional optimisation methods. They search from a *population of points*, not a single point. The search space is, therefore, 'attacked' at many points simultaneously, dramatically increasing the probability of reaching the global optimum. Furthermore, *GAs only need objective function information*. This enables them to be used in domains where continuity and derivative existence do not apply. They can also be used in 'noisy' search spaces, where the same set of parameters will not, in general, produce exactly the same result. Another characteristic of GAs is that they work with a *coding of the parameter set* and not with the parameters themselves. This makes them directly applicable to a wide range of non-numerical, combinatorial, and mixed optimisation problems. Finally, *GAs use probabilistic transition rules*, not deterministic ones, which adds to their generality and robustness. A more thorough comparison between evolutionary algorithms and classical gradient methods can be found in Salomon (1998) and the references therein.

For these and other reasons, GAs form a simple to implement, yet robust and generic optimisation tool, which can be used in many different classes of numerical as well as combinatorial optimisation problems. The truth of the above statements will become apparent in the following sections, where a mathematical framework for the analysis of the operation of simple GAs will be developed.

2.5 Mathematical Foundations of Genetic Algorithms

In the previous sections, genetic algorithms were introduced and their fundamental differences from conventional optimisers were outlined. Genetic algorithms can be thought of as crude models of natural genetics, employing principles such as the Darwinian 'survival of the fittest', resulting in a randomised, yet structured mechanism for the search of optimal solutions to optimisation problems. In Section 2.4 it was shown that a simple GA involves nothing more complex than random number generation, string copying, partial string exchanging and random bit alterations. Intuitively, based on the obvious success of natural genetics, one may expect GAs to perform well in most optimisation problems. Indeed, numerous applications of GAs to a wide range of domains have shown their superiority over conventional optimisation methods (see Holland, 1992; Fogel, 1994; Goldberg, 1989a, and the references therein). However,

despite their intuitive appeal and their experimental success, it is crucial that these positive indications about GAs are backed by rigorous mathematical facts. In order to accomplish this task, the important concept of *similarity templates* (or *schemata*) is introduced, which provides the notational tool necessary to arrive at an important theorem known as *The Fundamental Theorem of Genetic Algorithms* (also called the *Schema Theorem*). Most of the material in this section is due to Holland (1975), and De Jong (1975).

2.5.1 Similarity Templates (Schemata)

Recall that candidate solutions in a simple GA are represented by strings, which undergo reproduction, crossover, and mutation, in order to form new generations. To guide the search, highly fit strings (representing better solutions) are given a higher probability of surviving and propagating their genetic material through their offspring to successive generations. Instead of working with each string individually, it is often useful to group strings in the population based on their similarities. In order to accomplish this task, the concept of similarity templates (or schemata) is introduced.

Definition 2.1: A schema (Holland, 1975) is a similarity template describing a subset of strings with similarities at certain string positions. •

Without loss of generality, the discussion will be limited to strings of cardinality $k=2$. Specifically, the binary alphabet {0, 1} will be used. A schema is denoted by appending a new symbol to this alphabet. The symbol * is used to signify a 'don't care' string position. Hence, the schema **000 describes all strings of length $l=5$ whose three last positions are 0. Therefore, schema **000 describes the four distinct strings {00000, 01000, 10000, 11000}. In order to understand the importance of the notion of schemata, consider the problem of maximising the function $f(x)=x^2$. Assuming that direct binary coding is used to encode parameter x, it is obvious that strings containing 1 in their most significant bit will be fitter than other strings. Using the notion of schemata and assuming $l=5$, we can conclude that schema 1**** represents better solutions than schema 0****. Note that the sets represented by the two schemata are disjoint and their union forms the search space for the optimisation problem.

The notion of schemata helps us understand the amount of information that is contained within a single population in a GA. In terms of individual binary strings, the population can contain up to a total of 2^l different strings. However, in terms of schemata, due to the extended alphabet {0, 1, *} used, the population can contain up to a total of 3^l different schemata. Using a string length $l=5$, the upper bound in the number of schemata is $3^5=243$, as opposed to only $2^5=32$ in the case of individual strings. In general, for strings of cardinality k, there can exist k^l distinct strings and $(k+1)^l$ distinct schemata. It is now apparent that, by considering the strings, their fitness values, and the similarities among the strings in the population, there is a wealth of new information available to help direct the search.

The amount of information that is contained in a population is not constant and depends on the population *diversity* (i.e. the proportion of *distinct* strings in the population). A population containing copies of the same string is a worst-case example of a population with low diversity. Calculation of the precise amount of information contained in a population requires the explicit knowledge of the strings in the population. However, it is possible to establish upper and lower bounds on the total number of schemata in a population. It is easy to see that, in general, a particular string of length l is a member of 2^l schemata. This is because each string position can take its own value or the * symbol. Therefore, depending on the population diversity, a population of size N can contain somewhere between 2^l and $N \cdot 2^l$ schemata. Note that this result does not depend on the cardinality of the alphabet used to form the strings.

The above result clearly shows that even moderately sized populations can contain a wealth of information about the search space, which can be used to direct the search. The issue of whether GAs exploit this information or not still remains open. In order to show that GAs actually make use of the available information, the effects of the three fundamental genetic operators, reproduction, crossover, and mutation must be analysed in a systematic and rigorous way. This is the subject of the following section.

2.5.2 The Fundamental Theorem of Genetic Algorithms

In order to add rigour to the following discussion, some simple notation needs to be introduced. Without loss of generality, it is assumed that strings are constructed over the

binary alphabet {0, 1}. The state of a population at time (or generation) t is represented symbolically as $\boldsymbol{P}(t) = \{S_i(t), i = 1, 2, \ldots, N\}$, where the boldface denotes a population, $S_i(t)$ denotes the i-th string in the population at time t, and N denotes the population size. In order to improve clarity, the dependence of $S_i(t)$ on t may not be shown explicitly. Note that $\boldsymbol{P}(t)$ is a *multiset*, meaning that the elements of $\boldsymbol{P}(t)$ are not necessarily distinct.

A schema is simply a string constructed over the extended alphabet {0, 1, *}, and is represented by the letter H. An example of a schema is H = *10*0, which describes the four strings S_1 = 01000, S_2 = 01010, S_3 = 11000, and S_4 = 11010. The number of strings described by a schema is clearly a function of the number of '*' symbols contained in the schema. For example, schema H_1 = 110*0 describes much fewer strings than schema H_2 = 1****. Furthermore, schema H_1 spans more of the total string length than schema H_2. In order to quantify the above properties, the notions of schema *order* and *defining length* are introduced.

The order of a schema H is the number of fixed positions contained in a schema, and is denoted by $o(H)$. In the examples above, $o(H_1) = 4$, and $o(H_2) = 1$. The defining length of a schema H is the 'distance' between the first and last specific string position, and is denoted by $\delta(H)$. In the examples above, $\delta(H_1) = 5 - 1 = 4$, and $\delta(H_2) = 1 - 1 = 0$. In general, a schema H describes $k^{l-o(H)}$ distinct strings of cardinality k.

The effects of reproduction

In order to determine the effects of reproduction on the expected number of schemata in the population from time t to time $t+1$, suppose that, at time t, there are M_H strings that belong to a particular schema H within the population $\boldsymbol{P}(t)$. This is denoted as $M_H = M_H(t)$. Recall from Section 2.4.1 that during reproduction, a string S_i is copied according to its fitness value, denoted by f_i. In particular, the probability p_i of selecting string S_i for reproduction is given by

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2.1}$$

In a simple GA, the reproduction operator is applied N times resulting in a new set of N individuals that form a new population $\boldsymbol{P}'(t+1)$. Consider the M_H strings belonging to schema H. Obviously $M_H \leq N$, and without loss of generality, assume that the strings in the population are ordered in such a way that these M_H strings appear first in the sequence of N strings that form the population. Then, the probability of any one of these M_H strings propagating through to $\boldsymbol{P}'(t+1)$ is given by

$$p_r(H) = \sum_{i=1}^{M_H} p_i = \frac{\sum_{i=1}^{M_H} f_i}{\sum_{j=1}^{N} f_j} \tag{2.2}$$

Hence, since $\boldsymbol{P}'(t+1)$ is formed by selecting N strings from $\boldsymbol{P}(t)$ with replacement, the number of examples of schema H in $\boldsymbol{P}'(t+1)$ is given by

$$M'_H(t+1) = Np_r(H) = \frac{N\sum_{i=1}^{M_H} f_i}{\sum_{j=1}^{N} f_j} \tag{2.3}$$

Let $f(H)$ denote the average fitness of the strings representing schema H in population $\boldsymbol{P}(t)$, and $\bar{f}$ denote the average fitness of the entire population $\boldsymbol{P}(t)$.

$$f(H) = \frac{\sum_{i=1}^{M_H} f_i}{M_H(t)} \quad \text{and} \quad \bar{f} = \frac{\sum_{j=1}^{N} f_j}{N}$$

Combining Eq. (2.3) with the above equations leads to

$$M'_H(t+1) = M_H(t)\frac{Nf(H)}{\sum_{j=1}^{N} f_j} = M_H(t)\frac{f(H)}{\bar{f}} \tag{2.4}$$

Eq. (2.4) shows that the growth of a particular schema H can be expressed as the ratio of the average fitness of the schema to the average fitness of the entire population. In other words, schemata contained in population $\boldsymbol{P}(t)$ that have fitness values *above* the population average will receive an *increasing* number of string representatives in

population $\boldsymbol{P}'(t+1)$, whereas schemata contained in population $\boldsymbol{P}(t)$ that have fitness values *below* the population average will receive a *decreasing* number of string representatives in population $\boldsymbol{P}'(t+1)$. Furthermore, it can easily be shown that the schemata growth/decay rate is exponential.

The above results clearly show that GAs can successfully *exploit* the information contained in the population, by allocating exponentially increasing and decreasing numbers of schemata to subsequent generations. However, reproduction alone does not help the GA *explore* new regions in the search space, since it does not alter the genetic material contained in a given population. The mechanism of introducing new structures in the population, thus promoting exploration of the search space, is provided by the crossover and mutation operators.

The effects of crossover

Recall from Section 2.4.2 that the crossover operator involves the random mating and exchange of genetic material between pairs of selected strings. In single-point crossover, a crossing site is randomly chosen along the length of each string pair. Then, a new pair of strings is created by swapping all characters to the left or right of the crossing site (see Fig. 2.3). For strings of length l, there are $l-1$ possible crossing sites. This exchange of information clearly introduces new structures in the population, but may disrupt certain schemata, while leaving others unaffected. In particular, if the crossing site falls within the defining length δ of a given schema, this schema will be destroyed, whereas if the crossing site falls outside its defining length, the schema will survive. This means that schemata with short defining lengths have a higher probability of surviving crossover than schemata with long defining lengths. Assuming that the crossing site is uniformly chosen at random and that the crossover operator is applied with a probability p_c, it is possible to establish a lower bound on the crossover survival probability $p_c(H)$ of a schema H in terms of its defining length δ, which is given below.

$$p_c(H) \geq 1 - p_c \frac{\delta(H)}{l-1} \tag{2.5}$$

Consider $\boldsymbol{P}''(t+1)$, the population formed after the combined effects of reproduction and crossover. A lower bound on the expected number of strings representing schema H in

$\boldsymbol{P}''(t+1)$, denoted by $M''_H(t+1)$, can now be obtained by combining Equations (2.4) and (2.5) as shown below.

$$M''_H(t+1) \geq M_H(t)\frac{f(H)}{\bar{f}}\left[1-p_c\frac{\delta(H)}{l-1}\right] \tag{2.6}$$

The combined effect of reproduction and crossover can clearly be seen by examining the above equation. It is observed that schemata with both above-average fitness values and short defining lengths will propagate from population $\boldsymbol{P}(t)$ through to $\boldsymbol{P}''(t+1)$ and will receive samples at exponentially increasing rates.

The effects of mutation

Mutation is the last of the three fundamental genetic operators in a simple GA. In Section 2.4.3 the mutation operator was defined as the random alteration of a single position of a string, which occurs with a probability p_m. Hence, the survival probability of each individual character in a string after mutation is $1-p_m$. A schema is essentially defined by the values of the fixed positions within the schema. In order for a given schema to survive, all values of its fixed positions must be preserved intact. Since a schema H contains $o(H)$ fixed positions, and the individual mutations are statistically independent, a schema H survives mutation with a probability $p_m(H)=(1-p_m)^{o(H)}$. Similarly to natural population genetics, it is desirable for the mutation probability to be kept small in order not to severely disrupt the structures developed in the population. In this case, the schema survival probability mentioned above can be approximated by the expression $p_m(H) \approx 1-o(H)\,p_m$.

Consider $\boldsymbol{P}(t+1)$, the population formed after the combined effects of reproduction, crossover and mutation. A lower bound on the expected number of strings representing schema H in $\boldsymbol{P}(t+1)$, denoted by $M_H(t+1)$, can now be obtained by combining the above result with Eq. (2.6) as shown below (ignoring the cross-product term).

$$M_H(t+1) \geq M_H(t)\frac{f(H)}{\bar{f}}\left[1-p_c\frac{\delta(H)}{l-1}-o(H)p_m\right] \tag{2.7}$$

Since p_m is usually kept small, the contribution of the mutation operator does not alter much the earlier conclusion that highly fit schemata of short defining lengths receive

exponentially increasing samples in subsequent generations. Schemata possessing the above two properties are often called *building blocks*.

The results of Eq. (2.7) are central to the understanding of the underlying mechanism in the artificial evolution of a GA. They constitute the *Schema Theorem*, also known as the *Fundamental Theorem of Genetic Algorithms*. This theorem shows that GAs not only exploit the available information about a given search space, but also explore different regions of the search space in a manner that facilitates the creation of new and better structures. They accomplish this by using reproduction to exploit the available information about the search space, by segmenting it into smaller highly fit subsets represented by schemata with above-average fitness. Crossover is then used to further exploit and also explore the search space by combining building blocks, in order to form potentially better solutions. Finally, mutation promotes exploration by introducing new genetic material to the population, and guarantees the reachability of every point in the search space.

The interpretation of the Schema Theorem outlined above is known as the *building block hypothesis* (Goldberg, 1989a). Although this interpretation may seem too intuitive, there is a growing body of theoretical and empirical evidence to support it. For work on quantifying these ideas, the reader is referred to Holland (1975, 1987), Spears (1993), Thierens and Goldberg (1993), and Bethke (1981).

It should be noted here that the evolution process does not require information about the state of the population in past generations. The only information necessary for a simple GA to operate at any one time, is just the state of the current population of N strings. In fact, studies by Holland (1975) and Goldberg (1985) have shown that, despite the processing of only N structures in each generation, a GA effectively and usefully processes approximately N^3 schemata. Holland has given this result a special name, *implicit parallelism*. Among its important implications on information processing within a GA, implicit parallelism also facilitates the computer implementation of GAs, since the only memory requirement is the storage of the current state of the population of only N structures. Furthermore, a GA search run that was terminated at a given generation t can easily be resumed to continue the search, by simply initialising the population to its state at the terminating generation t.

2.6 Genetic Algorithm Implementation Issues

The performance of the Simple Genetic Algorithm (SGA) can be significantly improved by optimising certain elements in the algorithm, such as the sampling algorithm used in the reproduction operator, the encoding used to form the strings in the population, the population size, and the crossover and mutation probabilities. Some relevant theoretical and empirical results found in the GA literature are presented in the sections below.

2.6.1 Improving the Reproduction Operator

Consider $\boldsymbol{P}(t) = \{S_i(t), i = 1, 2, \ldots, N\}$, the multiset of all individuals in the population at time t. The reproduction phase begins by the determination of the expected number of samples of each individual to appear in $\boldsymbol{P}'(t+1)$, the population formed immediately after reproduction. Let s_i denote the expected number of samples of string S_i in $\boldsymbol{P}'(t+1)$. It clearly follows that

$$s_i = Np_i, \quad s_i \in \mathbb{R}_+ \tag{2.8}$$

where p_i is the probability of selecting string S_i for reproduction, obtained using Eq. (2.1). After computing the expected number of samples of each of the strings in $\boldsymbol{P}(t)$, a sampling algorithm is used to sample $\boldsymbol{P}(t)$ in order to form $\boldsymbol{P}'(t+1)$. For each of the strings in $\boldsymbol{P}(t)$, the sampling algorithm must be able to convert the corresponding expected number of samples s_i, a real number, to a discrete number of samples string S_i will actually receive in $\boldsymbol{P}'(t+1)$. The conversion must be made in such a way that the sampling is accurate and consistent. In order to quantify these goals, the measures of *sampling bias* and *sampling spread* are introduced.

Sampling bias

Sampling bias is defined as the absolute difference between an individual's actual, and expected number of samples. Let $\hat{s}_i$ denote the actual number of samples string S_i receives in $\boldsymbol{P}'(t+1)$. Then, the sampling bias can be written as $|s_i - \hat{s}_i|$. Sampling bias is a measure of sampling accuracy. The optimal value of sampling bias is zero and occurs whenever each individual's actual number of samples equals its expected number of samples.

Sampling spread

Sampling spread is defined as the range of possible values for $\hat{s}_i$. In the special case where $\hat{s}_i \in \{\lfloor s_i \rfloor, \lceil s_i \rceil\}$, the resulting spread is the smallest possible, which theoretically permits zero bias. This is called *minimum spread.* Sampling spread is a measure of sampling precision.

TABLE 2.1 Characteristics of commonly used sampling algorithms (Baker, 1987)

Sampling algorithm	Bias	Spread	Efficiency
Stochastic Sampling with Replacement (SSwR)	Zero (optimal)	Unlimited $0 \ldots N$	$O(N \log N)$
Stochastic Sampling with Partial Replacement (SSwPR)	Medium	Upper bounded $0 \ldots \lceil s_i \rceil$	$O(N \log N)$
Remainder Stochastic Sampling with Replacement (RSSwR)	Zero (optimal)	Lower bounded $\lfloor s_i \rfloor \ldots \lfloor s_i \rfloor + \Sigma s_i$	$O(N \log N)$
Remainder Stochastic Sampling without Replacement (RSSwoR)	Medium	Minimum $\lfloor s_i \rfloor, \lceil s_i \rceil$	$O(N \log N)$
Deterministic Sampling (DS)	High	Minimum $\lfloor s_i \rfloor, \lceil s_i \rceil$	$O(N \log N)$
Stochastic Universal Sampling (SUS)	Zero (optimal)	Minimum $\lfloor s_i \rfloor, \lceil s_i \rceil$	$O(N)$

Table 2.1 presents the basic characteristics of a number of commonly used sampling algorithms, in terms of sampling bias, sampling spread, and computational efficiency. It is observed that Stochastic Universal Sampling (SUS) outperforms all other sampling algorithms in both sampling bias and sampling spread. It is also more computationally efficient, delivering all N samples in a single pass. A visual representation of the SUS algorithm is shown in Fig. 2.5 below.

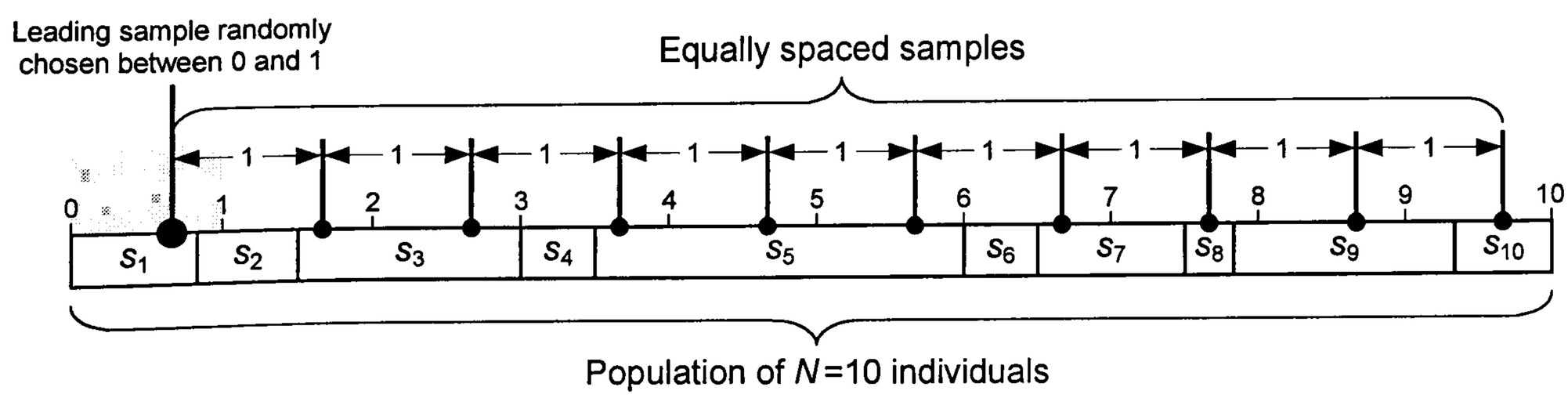


Fig. 2.5 *Example of the Stochastic Universal Sampling (SUS) algorithm*

In the beginning of the SUS algorithm, a leading sample is uniformly chosen at random between 0 and 1. The remaining $N-1$ samples are obtained by equally spacing them along the population 'line' of N unit segments, starting from the leading sample and maintaining a distance of 1 between successive samples, as shown in Fig. 2.5. This is equivalent to a gambler's spinning wheel, with the i-th wheel slice proportional in size to s_i, and with not only one, but N equally spaced pointers. In this regime, an individual is guaranteed to receive $\lfloor s_i \rfloor$ samples, but no more than $\lceil s_i \rceil$ samples. Hence, the SUS algorithm has minimum spread. Furthermore, in a randomly ordered population, the selection probability p_i, of a particular string S_i is only a function of the position of the leading sample (which is uniformly chosen at random) and the expected number of samples, s_i. Hence, the SUS algorithm has zero bias. In the example shown in Fig. 2.5, strings S_1, S_3, S_5, S_7, S_8, S_9, and S_{10} will receive 1, 2, 3, 1, 1, 1, and 1 copy in $\boldsymbol{P}'(t+1)$, respectively, whereas strings S_2, S_4, and S_6 will not appear in $\boldsymbol{P}'(t+1)$.

The optimal performance of the SUS algorithm makes it ideal for use in the reproduction operator, enabling the selection of individuals according to the theoretical specifications. More details and an empirical analysis of SUS and the other sampling algorithms shown in Table 2.1, can be found in Baker (1987).

2.6.2 Selection of String Encoding

The way in which candidate solutions are encoded into strings is of major importance for the success of a GA. In fact, the task of finding the best encoding for a given optimisation problem is equivalent to solving the problem itself (Mitchell, 1996). The suitability of a particular encoding depends on certain (and in most cases unknown) properties of the objective function associated with a given optimisation problem. Unfortunately, there are currently no rigorous guidelines for predicting which encoding will work best in a particular optimisation problem.

Binary string encoding

Although the mathematical framework developed earlier was based on strings constructed over the binary alphabet $\{0, 1\}$, the results obtained can be generalised to alphabets of arbitrary cardinality. However, there are certain theoretical justifications

for adopting the binary alphabet for the string encoding. Since a particular string of length l is a member of 2^l schemata, longer strings contain an exponentially increasing number of schemata, regardless of the cardinality of the alphabet used. For a particular candidate solution $x \in \mathbb{N}$, $x \neq 0$, the length of the encoded string under an alphabet of cardinality k is given by $l \approx \log_k x$. It is clear that the maximum possible string length is achieved by having $k=2$, resulting in the binary alphabet. This justification is also mentioned in Holland (1975).

Another reason for using binary encoding is the fact that the majority of theoretical and empirical studies of GAs and their properties, especially in terms of both rigorous and heuristic results about appropriate GA parameter settings, such as the crossover and mutation probabilities, are based on binary encoded strings, and cannot be easily generalised to alphabets of arbitrary cardinality.

Caruana and Schaffer (1988) proposed the use of *Gray coding* for the construction of strings. They argued that Gray coding usually results in more accurate solutions than other conventional codings, such as direct binary coding. This is attributed to the fact that, in Gray coding, adjacent integers differ by a single bit (a Hamming distance of 1). This adjacency property also results in smaller perturbations of the values of the strings under mutation. Practical applications, including results in this work, have indicated that Gray coding generally performs better that direct binary coding.

Adapting the string encoding

Based on the theoretical framework developed in Section 2.5, in order to improve the performance of a GA, the encoding used for constructing the strings must be such that functionally related, small string segments are more likely to stay together in a single string under crossover, in order to facilitate the building block hypothesis described earlier. However, this is not possible without knowing ahead of time which string segments are important in the formation of useful schemata. This is known in the GA literature as the *linkage problem*. Many scientists have tried to solve this problem by adapting the encoding used in a GA during the evolution process. The *inversion operator* developed by Holland (1975), works by reordering parts of strings, while preserving the functional interpretation of each of the string positions. Another

technique developed by Schaffer and Morishima (1987) works by evolving *crossover hot spots*, the positions at which crossover is allowed to occur, thus adaptively restricting 'dangerously disrupting' crossing sites from being used in the crossover operator. *Messy GAs* developed by Goldberg, Korb, and Deb (1989), attempt to improve the performance of GAs in function optimisation by building up increasingly longer and highly fit strings, derived from well-tested shorter building blocks. For more details on messy GAs, also refer to Goldberg, Deb, and Korb (1990), and Goldberg, Deb, Kargupta, and Harik (1993).

Selection of string length and parameter scaling

In most GA configurations, strings are formed by encoding each of the parameters involved in the optimisation, and concatenating the resulting string elements, as shown in Fig. 2.2. The length of each string element affects the resolution of the corresponding parameter representation. The longer the length of a string element, the more accurate the representation of the corresponding parameter becomes. However, since a string of length l and cardinality k results in a search space whose size is k^l points, long string elements can result in extremely large search spaces which can slow down the convergence of the GA.

In *linear scaling*, the difference between two successive numbers in a representation is given by $\Delta x = (x_{max} - x_{min})/(k^{l_x} - 1)$, where $x \in [x_{min}, x_{max}]$ and l_x is the length of the corresponding string element. Linear scaling results in a uniform distribution of points in the search space, thus maintaining a constant resolution throughout the search space. When the ranges of the parameters are not known, *logarithmic scaling* can be used, enabling a wider range of values to be searched using shorter strings. In logarithmic scaling, the k^{l_x} search points are logarithmically spaced in $[x_{min}, x_{max}]$, resulting in a distribution of points that is more dense in regions closer to x_{min}. This achieves a higher resolution in smaller parameter values, thus enabling shorter strings to be used more effectively. In cases where varying degrees of resolution are required at specific regions in the search space, *non-linear scaling* can be used where the required mapping can be described by any suitable, strictly monotonic, non-linear function. Fig. 2.6 illustrates three different types of parameter scaling.

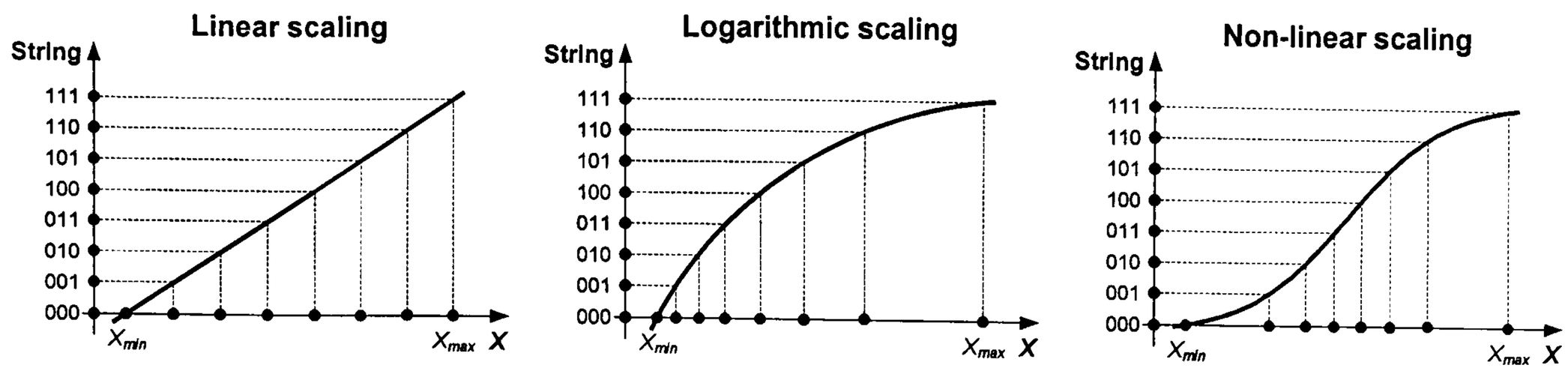


Fig. 2.6 Different types of parameter scaling with $k=2$ and $l=3$

2.6.3 Selection of Genetic Algorithm Parameters

In order to implement a genetic algorithm, the crossover and mutation probabilities and the population size need to be specified. Similarly to the problem of selecting a suitable string encoding, there are no rigorous guidelines for determining these parameters, as the optimal value of one parameter is, in general, a non-linear function of the values of the other parameters, thus making it impossible to optimise them one at a time. Fortunately, experience has shown that, in most optimisation problems, GAs are robust enough that the GA parameters do not severely affect their performance.

A number of researchers have investigated how the GA parameters affect the search performance of GAs. In particular, De Jong (1975) used a suite of test functions as a basis for testing the on-line and off-line performance of GAs with different combinations of GA parameters. On-line performance is computed by taking the average fitness of *all* individuals in all generations, whereas off-line performance is computed by taking the average of only the *best* individuals. In most function optimisation problems, it is the off-line performance that is of interest. De Jong found that both the on-line and the off-line performance of GAs peaked with a population size $N=50-100$ individuals, a single-point crossover probability $p_c \approx 0.6$, and a mutation probability $p_m \approx 0.001$ per string position.

Grefenstette (1986) performed a set of experiments in which a meta-level GA was used to optimise the parameters of other GAs that were set up to optimise the functions in De Jong's test suite. The parameters for the meta-level GA were set to De Jong's recommended values. The results of Grefenstette's work in terms of on-line

performance were $N=30$, $p_c=0.95$, and $p_m=0.01$. The generation gap discussed in Section 2.4.4 was also used in the experiments, and the optimal value was found to be $g=1$. In terms of off-line performance, the optimal values of the GA parameters were found to be $N=80, p_c=0.45, p_m=0.01$, and $g=0.9$.

In another experimental work, Schaffer, Caruana, Eshelman, and Das (1989) identified the optimal GA parameters for on-line performance to be $N=20-30$, $p_c=0.75-0.95$, and $p_m=0.005-0.01$, which are very similar to Grefenstette's results. The test suite used in this work was a set of numerical optimisation problems including some of De Jong's test functions, all encoded with Gray coding.

Heuristic formulas for optimal settings of p_c and p_m as functions of l and N have been proposed by Hesser and Männer (1991), where it was suggested that the optimal mutation probability, p_m, is inversely proportional to the string length, l. Loosely speaking, this is because with larger l the probability that good solutions are destroyed by mutation is higher, since longer strings are more likely to be mutated than shorter ones. The predicted GA parameters using the proposed formulas are in accordance with the experimental studies outlined above. Interesting theoretical results regarding the estimation of the optimal population size in a GA as a function of l have been published by Goldberg (1985), who showed that $N=1.65\times 2^{0.21l}$ for binary strings of length $l\leq 60$. Also refer to Goldberg (1989b), Ros (1989), and Goldberg, Deb, and Clark (1992, 1993) for further relevant results.

Note that care should be taken when treating the above recommendations for the GA parameters as globally optimal or universal. The results outlined above are applicable to the test suites used for conducting the experiments and cannot be easily generalised to every optimisation problem and string encoding. There is, however, a general trend emerging from the above results, suggesting a high crossover probability, a low (but non-zero) mutation probability, and a population size of 50 to 100 individuals. These settings can provide good initial values for the GA parameters, which can then be varied and fine-tuned as required by the particular optimisation problem.

Since the optimal GA parameters may change during the evolution process, many researchers have argued that there may be a need for the *adaptation* of the various GA

parameters as the genetic search is progressing. The reader is referred to the work by Davis (1989, 1991) for an interesting approach to the adaptation of the genetic operator probabilities, based on their observed performance. The basic idea for this approach was proposed much earlier by Cavicchio (1972). Relevant to the above discussion is the more recent work by Thierens (1995), in which he proposed an *adaptive crossover* algorithm. In this case, it is not the crossover probability that is adapted, but the crossover algorithm itself.

2.6.4 Incorporating Problem Specific Knowledge

In many optimisation problems, useful information about the problem may already exist, and can often be used *a priori* to effectively help GAs perform better in terms of both *rate of convergence* and *solution accuracy*. Prior knowledge can be incorporated in genetic search in various ways, the most commonly used being the careful initialisation of part of the population at time $t = 0$, and the combination of GAs with local optimisers such as hill-climbing algorithms.

Improving the rate of convergence

If there exists prior information about regions in the search space where the optimal points may lie, a percentage of the population at time $t = 0$ can be initialised by selecting candidate solutions from these promising regions. This approach can be used whenever one seeks to improve on previously computed 'optimal' solutions. In this way, the GA begins with a set of potentially above-average solutions, which can significantly improve the rate of convergence of the GA. This approach has been applied by Grefenstette (1987) to the solution of the travelling salesman problem. While the crossover and mutation operators theoretically ensure that the algorithm will still explore different regions in the search space, such heuristic initialisations of the population should be applied carefully, in order to avoid premature convergence, the situation where the GA converges to a sub-optimal region in the search space.

Improving the solution accuracy

Another approach is to 'refine' the results obtained by the GA, using hill-climbing or any other suitable conventional optimisation algorithm. For example, if there is *a priori*

knowledge that the search space is locally continuous and differentiable, but highly multimodal, a GA can be used to locate the optimal 'peak', while a hill-climbing algorithm can be used to focus on the identified, locally unimodal region of the search space. Such combinations of GAs and conventional optimisers are often termed *hybrid GAs*. There are many application examples in the GA literature demonstrating the improved performance of hybrid GAs over simple GAs. For a few such examples, refer to Hart and Belew (1996), and Salomon (1998).

2.6.5 Genetic Algorithm Termination

Due to the stochastic nature of GAs, conventional termination criteria such as the ones based on the precision of the obtained solutions, cannot be directly applied to GAs. In most practical GA implementations, the algorithm is terminated after a given number of generations, and the best individuals in the final generation are assessed. If the resulting solutions are not satisfactory, the GA can be restarted, or a new GA can be initiated, in which part of the population at time $t = 0$ is initialised with the best individuals found in previous GA search runs (see Section 2.6.4).

2.7 Multiobjective Optimisation and Genetic Algorithms

Multiobjective or multicriteria optimisation problems arise in cases where several, often conflicting objectives are present, thus resulting in vector-valued objective functions. In fact, the majority of optimisation problems are multiobjective in nature, but are usually treated by weighting and combining all objectives into a single-valued function, thus transforming them into single-objective optimisation problems. This approach may be acceptable in certain cases, but there are times when combining the objectives in an efficient way may not be practically feasible.

A multiobjective optimisation problem is equivalent to the problem of simultaneously optimising the m elements of a vector-valued function, such as the one shown below.

$$\mathbf{f}(\mathbf{x}) = \left[f_1(\mathbf{x}) \ \cdots \ f_m(\mathbf{x}) \right] \in \mathbb{V} \tag{2.9}$$

where $\mathbf{x} = \left[x_1 \ \cdots \ x_n \right] \in \mathbb{U}$ is the n-dimensional parameter vector. Sets $\mathbb{U}$ and $\mathbb{V}$ are the effective domain and codomain of $\mathbf{f}(\mathbf{x})$, respectively, as defined in the context of the

given optimisation problem. Since the objectives $f_i(\mathbf{x}), i \in \{1, \ldots, m\}$ are all functions of parameter vector $\mathbf{x}$, they cannot be optimised one at a time. It is, therefore, clear that the notion of optimality in the context of multiobjective optimisation must be redefined in a way that respects the integrity of each objective, and at the same time permits interactions between objectives. Precisely these goals are achieved by the concept of *Pareto-optimality*. A parameter vector $\mathbf{x}$ is said to be Pareto-optimal if and only if there is no other parameter vector which can improve any of the individual objectives, $f_i(\mathbf{x})$, without worsening at least one other objective. In order to make the above definition mathematically more rigorous, the following additional definitions are necessary.

Definition 2.2: Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$. Vector $\mathbf{x}$ is *partially less* than $\mathbf{y}$, denoted as $\mathbf{x} \lesssim \mathbf{y}$, if and only if

$$\forall i \in \{1, \ldots, m\}, \quad x_i \le y_i \ \wedge \ \exists\, i \in \{1, \ldots, m\} \mid x_i < y_i \qquad \bullet$$

Definition 2.3: Assuming a minimisation problem, parameter vector $\mathbf{x}_1 \in \mathbb{U}$ belongs to the Pareto-optimal set of function $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^m$ as defined in (2.9), if and only if there is no other parameter vector $\mathbf{x}_2 \in \mathbb{U}$ such that $\mathbf{f}(\mathbf{x}_2) \lesssim \mathbf{f}(\mathbf{x}_1)$. •

Vectors belonging to the Pareto-optimal set are called *non-dominated* or *non-inferior*, while all remaining vectors are called *dominated* or *inferior*. In order to better illustrate the concept of Pareto-optimality, consider the following vector-valued function.

$$\mathbf{f}(x) = \left[x^2 \ \ (x-2)^2\right] \tag{2.10}$$

Fig. 2.7 illustrates the dominated and non-dominated regions in the solution plane for the above unidimensional function. It is observed that the non-dominated region consists of not only one solution but a *family of solutions* that form the Pareto-optimal set. All points in this set are, in general, optimal solutions of *equal importance* and a good multiobjective optimiser should be able to locate all points in the set. It is interesting to note that the conventional, weighted-sum approach of combining all weighted individual objectives into a singe-valued function, such as

$$J(\mathbf{x}) = \sum_{i=1}^{m} w_i f_i(\mathbf{x}), \quad w_i \ge 0 \tag{2.11}$$

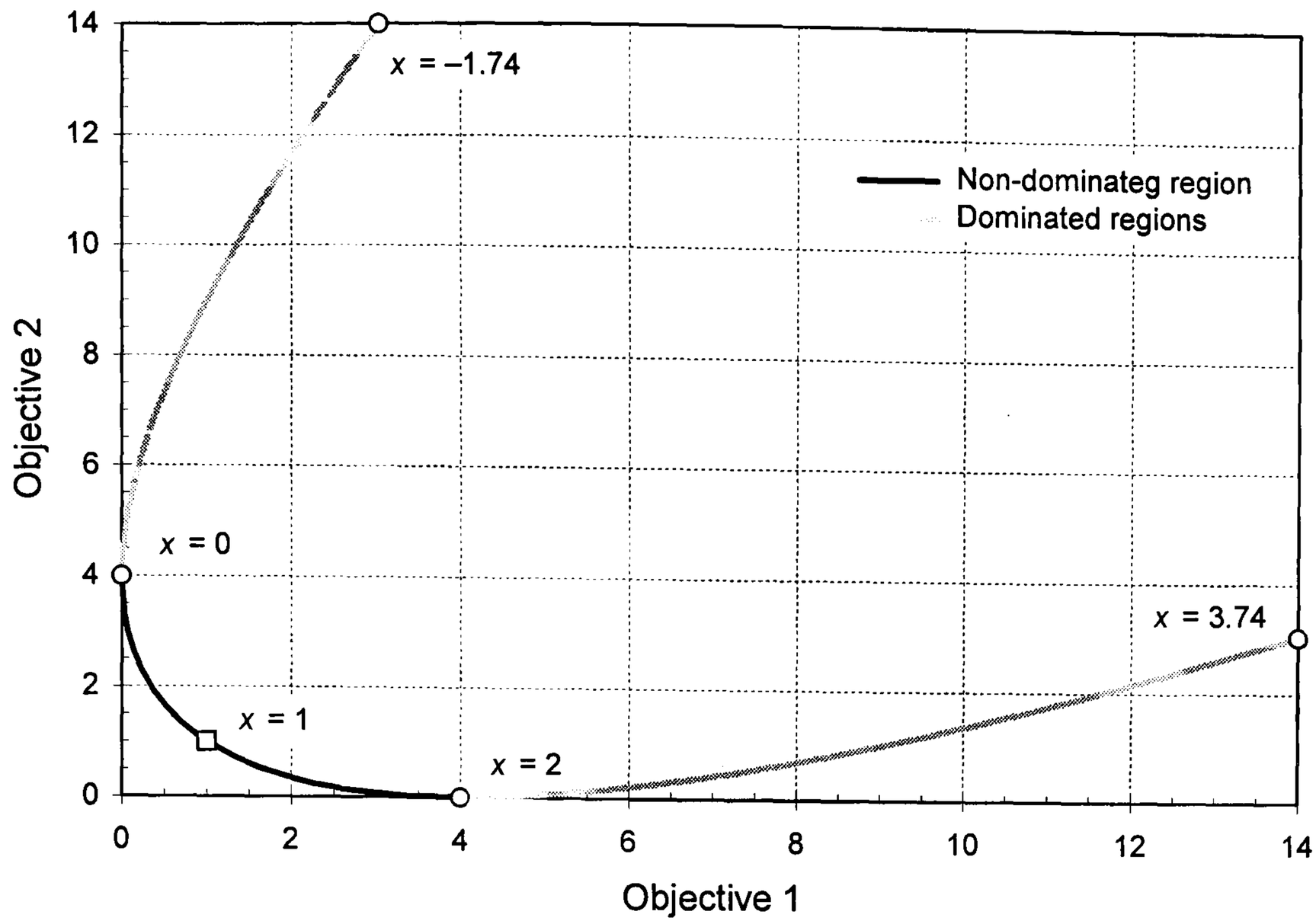


Fig. 2.7 *A simple multiobjective problem (Schaffer, 1985)*

results in a *single* Pareto-optimal solution. The location of this solution within the non-dominated region is determined by the choice of the weighting coefficients w_i. As an example, using objective function (2.11) to minimise the elements of function (2.10) results in the following expression for the optimal value of parameter x.

$$x_{opt} = \frac{2w_2}{w_1 + w_2}$$

Hence, when $w_1 = w_2 = 1$, objective function (2.11) is minimised at the Pareto-optimal point $x = 1$ (see Fig. 2.7). Varying the weighting coefficients results in different unique solutions spanning the Pareto-optimal region $x \in [0, 2]$.

In most real-world optimisation problems, the relationship between the optimal values of weighted-sum objective functions such as (2.11) and the weighting coefficients w_i is complex and unknown *a priori*, and it is not at all clear how one should weight the various objectives in order to arrive at a specific Pareto-optimal solution. This practically unpredictable bias towards specific solutions is the main weakness of conventional, weighted-sum approaches to multiobjective optimisation. More details on

multiobjective optimisation and an overview of different methods and applications can be found in Hwang and Masud (1979).

Multiobjective genetic algorithms

It is clear from the preceding discussion that multiobjective optimisation problems must be treated in a way that respects the definition of Pareto-optimality. The existence of multiple optimal solutions, even in such simple multiobjective problems as the one illustrated in Fig. 2.7, results in inherently multimodal search landscapes, and this is true even in cases where the individual objectives are all unimodal and convex functions of the decision variables. This property alone can render most conventional optimisation techniques inapplicable, as they were not designed to handle multiple solutions.

Although the parallel evolution of multiple individuals makes GAs structurally superior to most conventional search techniques in the solution of multiobjective optimisation problems, the inherently scalar way in which GAs process fitness information makes them unsuitable for such problems in their standard form. Many researchers have attempted to modify standard GAs, in order to make them applicable to multiobjective optimisation, with various degrees of success. The following is only intended as a brief review of the work that has been done in this field. A more complete review can be found in Fonseca and Fleming (1995).

The first significant contribution in this direction is the work by Schaffer (1985), who developed what is known as the *Vector Evaluated Genetic Algorithm (VEGA)*. In his approach, the population consists of a number of equally sized sub-populations, each one corresponding to a particular objective. The reproduction operator is applied locally in each sub-population, resulting in individuals being selected based on their performance on the corresponding objective only. The crossover operator, however, is applied to individuals across sub-population boundaries, thus enabling the mixing of individuals corresponding to different objectives. Non-dominated individuals are identified as the population evolves, but this information is not used by the VEGA itself. Although this scheme is intuitive and easy to implement, in can be shown (Richardson, Palmer, Liepins, and Hilliard, 1989) that it results in a bias against certain members of the Pareto-optimal set, something against the notion of Pareto-optimality.

A different approach was proposed by Goldberg (1989a), in which each individual in the population is *ranked* on the basis of non-dominance. All non-dominated individuals in the current population are identified and assigned a rank of 1. Among the individuals that have not yet been ranked, the non-dominated ones are identified and assigned a rank of 2. The process is repeated until all individuals in the current population are ranked. The reproduction operator is then applied based on the rank of each individual. In this way, all non-dominated individuals (i.e. all members of the Pareto-optimal set) are guaranteed to receive an equal probability of reproduction. This rank-based approach theoretically enables the GA to locate all points in the Pareto-optimal set. However, in order to maintain a sufficient degree of diversity among individuals, this approach should be used in conjunction with the more advanced techniques of *niche formation* and *speciation*. These techniques enable a more even distribution of the members of the population along equally important regions in the search space. More details on niche formation and speciation can be found in Chapter 4. An introduction to this topic can be found in Goldberg (1989a) and the relevant references therein.

More recently, Fonseca and Fleming (1993, 1998a, 1998b) proposed a Pareto-based method known as the *Multiobjective Genetic Algorithm (MOGA)*, which is an extension of the Pareto-based method proposed by Goldberg (1989a) and outlined above. In this approach, each individual is ranked according to the number of individuals in the current population by which it is dominated. In this way, non-dominated individuals are all assigned the same rank, while dominated ones are penalised according to the population density in the corresponding region of the trade-off surface. Another important element of the MOGA approach is the so-called *Decision Maker (DM)*, which is used to direct the search towards the most promising regions in the search space by combining dominance with partial preference information. In the beginning of the evolution process, the DM uses the *a priori* domain-specific knowledge about the problem (if such knowledge exists), to produce a suitable fitness assignment strategy for the members of the currently identified Pareto-optimal set. As the artificial evolution progresses, the acquired knowledge obtained by the GA is used by the DM to refine its fitness assignment strategy accordingly. In this way, the algorithm can effectively identify a subset of the Pareto-optimal set that satisfies certain auxiliary optimality criteria. The DM may range from a conventional, weighted-sum approach, to an intelligent decision maker operating manually.

2.8 Genetic Algorithms as Universal Optimisers

The preceding discussion may have given readers the impression that genetic algorithms outperform other optimisation algorithms on virtually all classes of optimisation problems. Unfortunately, recent studies by Wolpert and Macready (1997) revealed that this is not the case. Their central result is an important set of theorems known as the *No Free Lunch Theorems (NFL)*. The implications of these theorems are general, and are not limited to GAs. Loosely speaking, the NFL theorems state that there can never exist any one optimisation algorithm that works better than any other, when averaged over the set of *all* possible optimisation problems. In particular, if algorithm A outperforms algorithm B on some cost functions, there must exist exactly as many other functions where B outperforms A. Put in other terms, no optimisation algorithm can ever exist that is better than random search (or any other algorithm), if its performance measure is averaged over *all* possible cost functions. The NFL theorems clearly imply that *a priori* domain-specific knowledge must be incorporated in a given search algorithm, in order for it to perform well in that particular domain. More relevant results in this direction can be found in Macready and Wolpert (1996, 1998), and Culberson (1998).

The NFL theorems and their implications raise important questions regarding GAs and the justification for their preference over conventional optimisers. Since standard GAs are essentially blind search algorithms, how can one explain their success in numerous practical applications? A possible explanation may be that the majority of cost functions encountered in practice exhibit certain common properties which facilitate the use of GAs in some way. Note that the set of "all possible cost functions" used in the statement of the NFL theorems, certainly contains an infinite number of 'unusual' functions not likely to be encountered in practice. Although the explanation given above has an intuitive appeal, it should be treated carefully, and blind faith in a particular algorithm, including GAs, to search effectively and efficiently across a broad class of problems should, in general, be avoided.

2.9 Summary

In this chapter, genetic algorithms (GAs) as function optimisers were introduced, and their fundamental differences with conventional optimisation algorithms were outlined.

A brief introduction to the structure and operation of simple GAs was given, followed by a rigorous mathematical treatment, including the important concept of a *schema*, the *building block hypothesis*, and the *fundamental theorem of genetic algorithms*. Certain important issues regarding the implementation of GAs were then presented, including guidelines for selecting various parameters, such as the string encoding, the population size, the crossover and mutation probabilities, and others. Issues regarding incorporating problem specific knowledge in GAs were discussed, followed by a brief introduction to multiobjective optimisation using GAs. Finally, the role of GAs as universal optimisers was briefly discussed, including the important *no free lunch theorems* for optimisation. The material contained in this chapter is standard, and is only intended as a brief review of the current state of affairs in the field of GAs as function optimisers.

3 Analysis and Design of Objective Functions for Control Systems

3.1 Introduction

This chapter is primarily concerned with the analysis and design of objective functions, as applied to the solution of control engineering problems. A number of conventional objective functions are introduced, and it is shown that they can often provide poor controller performance indexes. A novel objective function is then proposed for the solution of control problems involving single-input, single-output (SISO) processes, that overcomes many of the weaknesses of conventional objective functions. The proposed objective function is experimentally analysed using a simple control problem, involving the tuning of a proportional plus integral (PI) controller for a linear, time-invariant process with time delay. The obtained results are analysed and compared with those obtained using conventional objective functions, as well as using several conventional PI controller tuning methods. Finally, the proposed objective function is generalised to multi-input, multi-output (MIMO) control problems.

3.2 Conventional Objective Functions

Consider the general, closed-loop SISO system shown in Fig. 3.1.

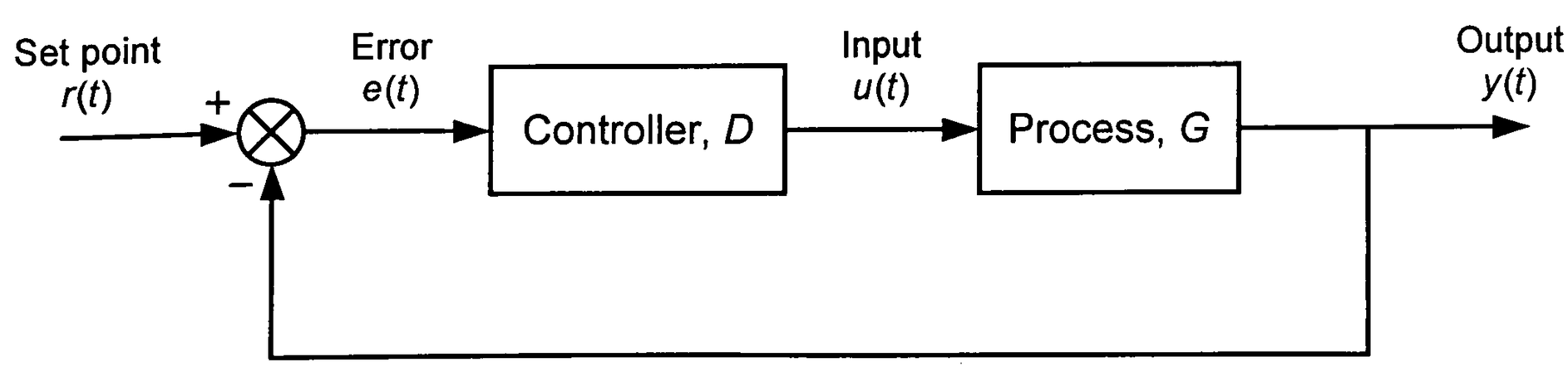


Fig. 3.1 General, closed-loop SISO system

In the system shown in Fig. 3.1, the controller and process can be linear or non-linear. The performance of the controller, D, in this system can be assessed by examining the signals $r(t)$, $e(t)$, $u(t)$, and $y(t)$. This can be expressed mathematically as

$$J(D) = f\left[r(t), e(t), u(t), y(t)\right] \in \mathbb{R}, \quad D \in \mathcal{D}, \quad 0 \le t < \infty \tag{3.1}$$

where $J(\cdot)$ is a single-valued function, often called *objective function*, and $\mathcal{D}$ is the set of all permissible controllers. Each controller in $\mathcal{D}$ is also known as a *solution* to the control problem. It is common practice to design J in such a way that the optimal controller, D_{opt}, is the one at which J attains its minimum value. Hence, D_{opt} is obtained by solving the following minimisation problem.

$$J\left(D_{opt}\right) = \min J(\mathcal{D}) \tag{3.2}$$

A suitable and simple requirement is that the error signal $e(t)$ should be kept as small as possible, and $e(t) \to 0$ as $t \to \infty$. A number of objective functions that are based on this requirement are shown below.

$$\begin{aligned} J_{IAE}(D) &= \int_0^\infty |e(t)|\, dt & J_{ITAE}(D) &= \int_0^\infty t\,|e(t)|\, dt \\ J_{ISE}(D) &= \int_0^\infty e^2(t)\, dt & J_{ITSE}(D) &= \int_0^\infty te^2(t)\, dt \end{aligned} \tag{3.3}$$

In the *Integrated Absolute Error (IAE)* and *Integrated Squared Error (ISE)* objective functions, the entire error signal is assigned the same weight, whereas in the *Integrated Time and Absolute Error (ITAE)* and *Integrated Time and Squared Error (ITSE)*, the weight on the error starts from zero at $t = 0$ and increases linearly with time, in order to allow for the transient error signal which will always be large.

Using any one of the objective functions in (3.3), the control problem in Fig. 3.1 can be transformed into a function optimisation problem, where the objective function J is minimised over $\mathcal{D}$. Consider the case where the process is controlled by a linear controller of the general form

$$D(s) = \frac{b_0 s^m + b_1 s^{m-1} + \cdots + b_{m-1} s + b_m}{s^k (s^n + a_1 s^{n-1} + \cdots + a_{n-1} s + a_n)} e^{-sL} \tag{3.4}$$

Then, the objective function would be of the form

$$J(a_1,\ldots,a_n,b_0,\ldots,b_n,L,m,n,k) = f\left[r(t),e(t),u(t),y(t)\right], \quad 0 \le t < \infty \tag{3.5}$$

Note that, in this case, the minimisation of J is a mixed optimisation problem, in which both the *parameters* (real variables, a_i, b_i, L) and the *structure* (integer variables, m, n, k) of the controller $D(s)$ are tuneable. The discontinuities associated with the integer variables make this problem extremely difficult to solve numerically using conventional optimisation techniques. GAs, however, can be directly applied to such problems by simply defining a suitable chromosome structure, as described in Chapter 2.

3.2.1 Limitations of Conventional Objective Functions

The performance specifications for a given control problem are often associated with the shape of the closed-loop system transient response to some set point signal such as a step or ramp function. Standard response characteristics such as peak overshoot, settling time, rise time, steady-state offset, velocity lag, and others may be used to specify a required controller performance. In order to provide reliable performance indexes, the objective function should, therefore, be able to adequately describe such specifications.

In the case of objective functions such as the ones shown in (3.3), it is clear that the simple error minimisation requirement may not be adequate for the description of such specifications. In general, this can result in mathematically optimal solutions that may be unacceptable in practice, as illustrated in Fig. 3.2.

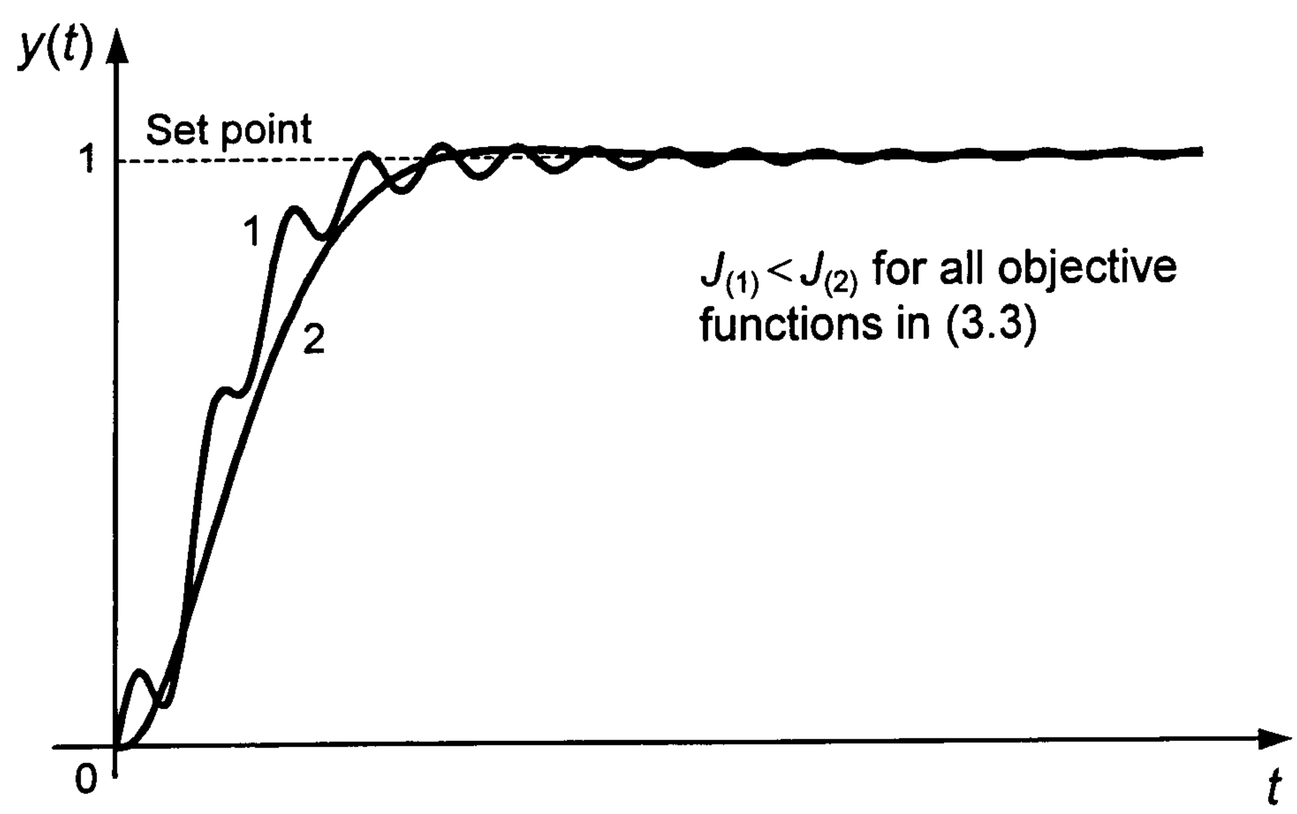


Fig. 3.2 Closed-loop system responses

With respect to Fig. 3.2, all objective functions shown in (3.3) falsely indicate that the controller associated with the oscillatory response 1 performs better than that associated with the almost critically damped response 2. This illustrates that a mere minimisation of the error signal can lead to qualitatively unacceptable solutions. The occurrence of this problem can be minimised by adding an extra term in the objective function, that penalises excessive variations in the process input signal, $u(t)$. This approach, however, can result in relatively conservative controllers.

The performance of the objective functions in (3.3) can be improved by minimising the error signal that results by taking the difference between the actual system response and a *desired* system response that meets the design specifications. However, a solution that precisely achieves the desired system response may not exist, especially in cases where the structure of the controller remains fixed. Therefore, similarly to the case illustrated in Fig. 3.2, an attempt to minimise the objective function may result in system responses that 'follow' the desired response but are qualitatively unacceptable.

Another disadvantage of the objective functions in (3.3) is that they almost always result in search landscapes where the global minimum is achieved by a *unique* solution, even though an infinite number of solutions may exist that satisfy the given specifications. A well-designed objective function should be able to identify all solutions that completely satisfy the specifications, without bias towards specific solutions. This enables the designer to manually examine the obtained solutions, and choose one which also satisfies certain qualitative objectives which cannot easily be expressed in mathematical terms. This is generally not possible using the objective functions in (3.3).

3.3 A Novel Objective Function for Control Systems

In this work, a novel objective function has been designed for use in control engineering problems, that overcomes most of the limitations of conventional objective functions outlined earlier. The main advantage of the proposed objective function is that it enables the designer to explicitly specify the required performance specifications for a given problem, in terms of time-domain bounds on the closed-loop system responses. The proposed objective function is initially formulated for use in problems involving SISO processes, but can easily be generalised to MIMO control problems.

The proposed objective function was designed in order to satisfy the following three fundamental requirements.

1. It must allow arbitrary performance specifications to be given in the time-domain, in a precise and straightforward manner.
2. All solutions that satisfy the specifications must be mapped to a single number.
3. It must be applicable to a wide range of linear, non-linear, and time-varying control systems of arbitrary structures.

Requirement 1 is based on the fact that the majority of performance specifications are expressed in terms of transient characteristics of the closed-loop system, such as peak overshoot and settling time. Furthermore, time-domain specifications are generally better understood by control personnel. In the case where many solutions that satisfy the specifications exist, requirement 2 ensures that there will be no bias towards specific solutions. Finally, requirement 3 enables the objective function to be used in complex control problems involving non-linear and time-varying elements, properties that are very common in practice.

3.3.1 Objective Function Formulation

Consider the general SISO system shown in Fig. 3.1. The set of desired time-domain response characteristics can be thought of as an area in the (y, t) plane. One such area is shown as the shaded region in Fig. 3.3, where the desired response characteristics may be expressed in terms of the constants c_1, c_2, c_3, c_{ss}, and t_2, as follows.

Peak overshoot: $\leq \left(\dfrac{c_1 - c_{ss}}{c_{ss}}\right) \times 100\%$

Settling time: $\leq t_2$, where the settling time is $\max\left\{\dfrac{c_2 - c_{ss}}{c_{ss}}, \dfrac{c_{ss} - c_3}{c_{ss}}\right\} \times 100\%$

Every response whose trajectory lies entirely within the shaded region will have the above characteristics. Note, however, that the converse is not true, mainly because of the additional specifications imposed by c_4 and t_1. Let J_S denote the area that is formed by the parts of the response curve that *do not* belong to the shaded region. Area J_S is the sum of the areas marked ▨ in Fig. 3.3. The magnitude of J_S gives an indication of how

close the system performance is to the desired for a particular controller, and hence a measure of the ability of the controller to meet the specifications. The larger the magnitude of J_S, the poorer the controller performance, with a value of zero indicating that the performance specifications have been completely satisfied.

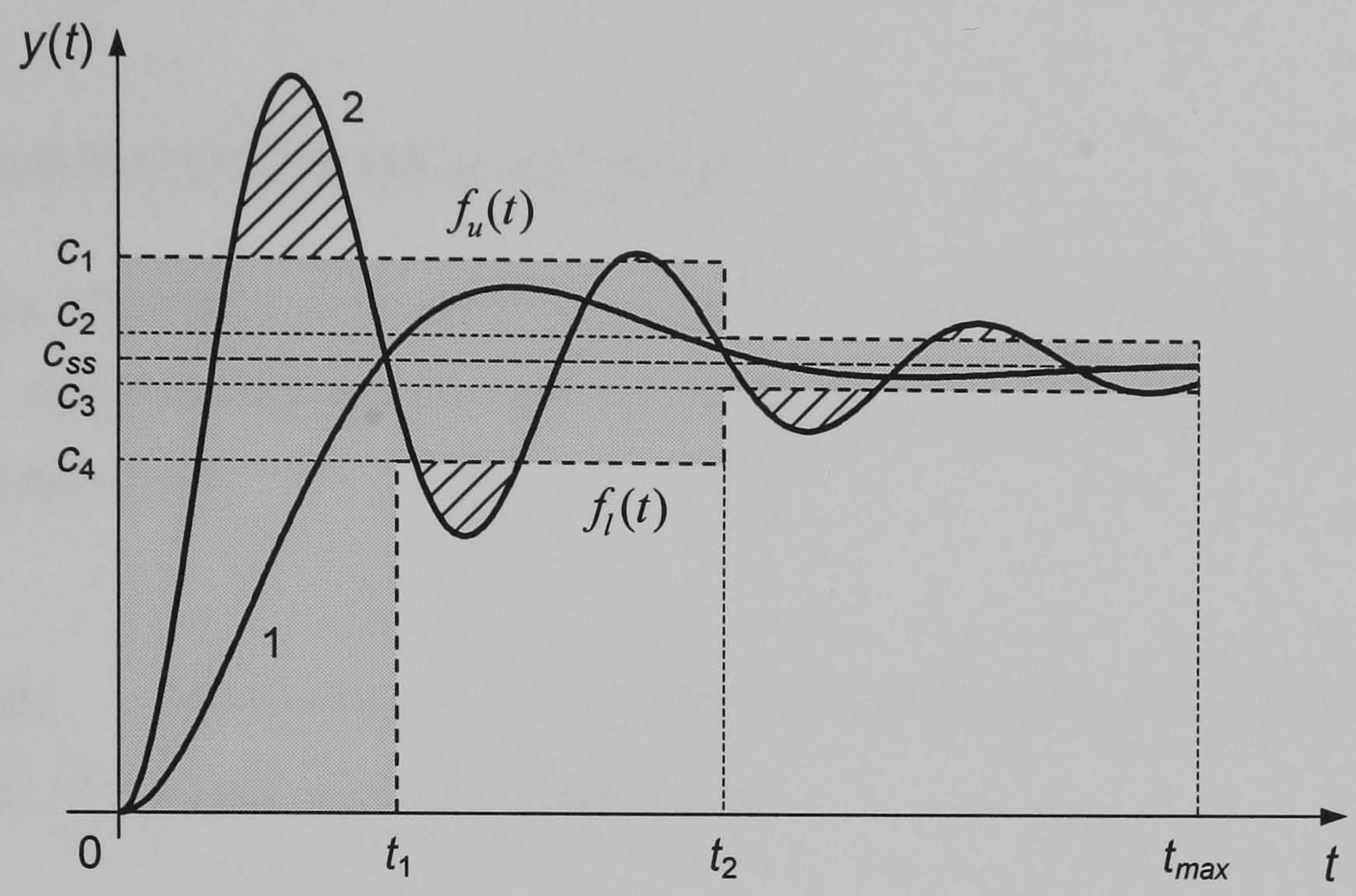


Fig. 3.3 *Typical set point tracking performance specifications*

With reference to Fig. 3.3, the controller that produces response curve 1 has totally met the specifications. On the other hand, parts of response curve 2 lie outside the shaded region. The value of the objective function is defined as the magnitude of area J_S in the time range $[0, t_{max}]$. Hence, for response 1 the value of the objective function is zero, whereas for response 2 it is a positive real number. Therefore, J_S can be thought of as a measure of the 'distance' between the candidate solution and the set of solutions that completely satisfy the specifications.

Note that the region representing the desired specifications need not be of the same shape as the one shown in Fig. 3.3. Any number of response constraints within the simulation time range $[0, t_{max}]$ can be arbitrarily defined by the designer, thus making the proposed objective function extremely flexible. In mathematical form, the proposed objective function is defined as

$$J_S(D) := \int_0^{t_{max}} \left(\max\{f_l(t) - y(t), 0\} + \max\{y(t) - f_u(t), 0\}\right) dt \tag{3.6}$$

where $D \in \mathcal{D}$ is the candidate controller, and $f_l(t)$ and $f_u(t)$ are user-specified functions defining the lower and upper boundaries, respectively, of the region representing the performance specifications, with $f_l(t) \leq f_u(t)$ for all t in $[0, t_{max}]$. In objective function (3.6), the integration is not carried through infinity, but truncated up to t_{max}, the time required for the system to reach the steady state.

3.3.2 Optimal and Strictly Optimal Solutions

The following definitions are necessary in order to improve clarity, by distinguishing between the different degrees of optimality that can be achieved when minimising objective function (3.6).

Definition 3.1: With reference to the objective function (3.6), a solution D in $\mathcal{D}$ is called *optimal*, if and only if $J_S(D) = \min J_S(\mathcal{D})$. •

Definition 3.2: With reference to the objective function (3.6), a solution D in $\mathcal{D}$ is called *strictly optimal*, if and only if it belongs to the kernel of J_S. The set of all strictly optimal solutions is, therefore, $\tilde{\mathcal{D}} = \{D \in \mathcal{D} \mid J_S(D) = 0\}$. •

From the definitions above, it is clear that optimal solutions will always exist, whereas strictly optimal solutions may or may not exist, depending on the given specifications and the capabilities of the control scheme employed. The fact that $J_S : \mathcal{D} \mapsto \mathbb{R}_+$ ensures that all strictly optimal solutions are also optimal, and that all strictly optimal solutions are favoured equally, since they are all assigned the same performance index of zero.

The analytical minimisation of J_S can be a very difficult task, even in cases where the process and controller are both linear and time-invariant. However, the development of fast computers and powerful control system simulation packages has made *numerical* optimisation a feasible and attractive alternative to analytical optimisation. This enables complex objective functions such as (3.6) to be optimised numerically, an approach that has the potential to deliver excellent designs that go beyond conventional, linear, time-invariant control theory. If a model of the controlled process exists, J_S can easily

be evaluated for a particular candidate controller by simulating the resulting closed-loop system and evaluating the integral in (3.6) numerically. This makes the proposed objective function applicable to a wide range of linear, non-linear and time-varying closed-loop systems.

3.4 Experimental Analysis

In this section, the objective function developed earlier is experimentally analysed, by applying it to a simple parametric controller design problem involving a SISO, linear, time-invariant process with time delay. The obtained results are analysed and compared with those obtained using the four conventional objective functions shown in (3.3), as well as a set of five well-known, empirical controller tuning rules.

The objective is to tune a Proportional plus Integral (PI) controller, so that the resulting closed-loop system meets some given performance specifications. The closed-loop system has the following standard configuration.

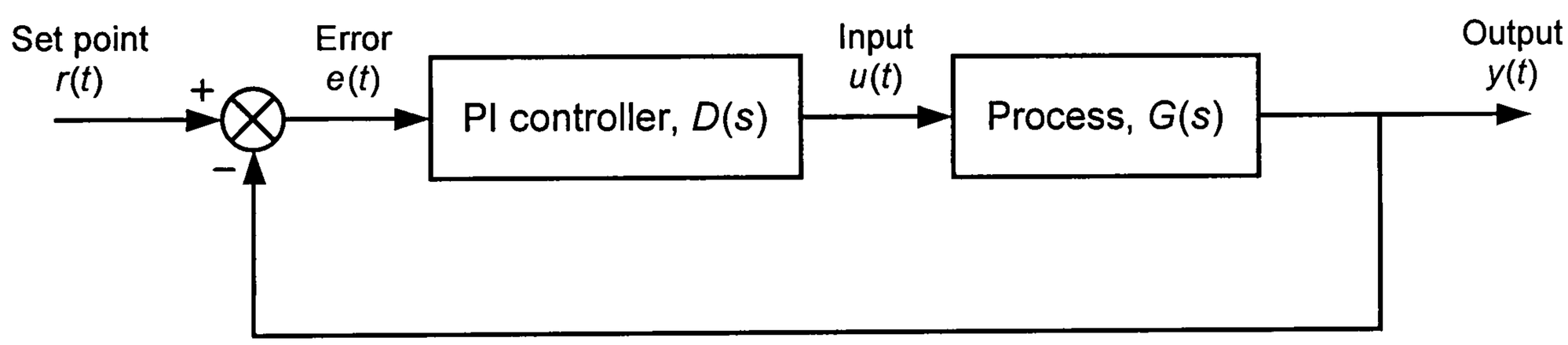


Fig. 3.4 *A simple parametric controller design problem*

In the system shown in Fig. 3.4, the transfer functions of the process and controller have the following structure.

$$G(s) = \frac{K}{Ts+1}e^{-sL} \quad \text{and} \quad D(s) = K_p + \frac{K_i}{s} \tag{3.7}$$

It is observed that the process is described by a standard first-order dead time model, with $K=0.5$, $T=10$ sec, and $L=3$ sec. Controller parameters K_p and K_i are unknown. The reason for choosing this particular PI controller structure will become apparent later

LIVERPOOL
JOHN MOORES UNIVERSITY
AVRIL ROBARTS LRC
TEL. 0151 231 4022

in this section, where it will be shown that this controller structure enables the parameterisation of all stabilising PI controllers in a way that facilitates the graphical determination of the relative stability and robustness properties of the solutions, based on their location in the parameter plane (K_p, K_i).

3.4.1 The Performance Specifications

The performance specifications for the control problem were given as time-domain bounds in the (y, t) plane, of the response of the closed-loop system to a unit step function. They are represented by the shaded region shown in Fig. 3.5.

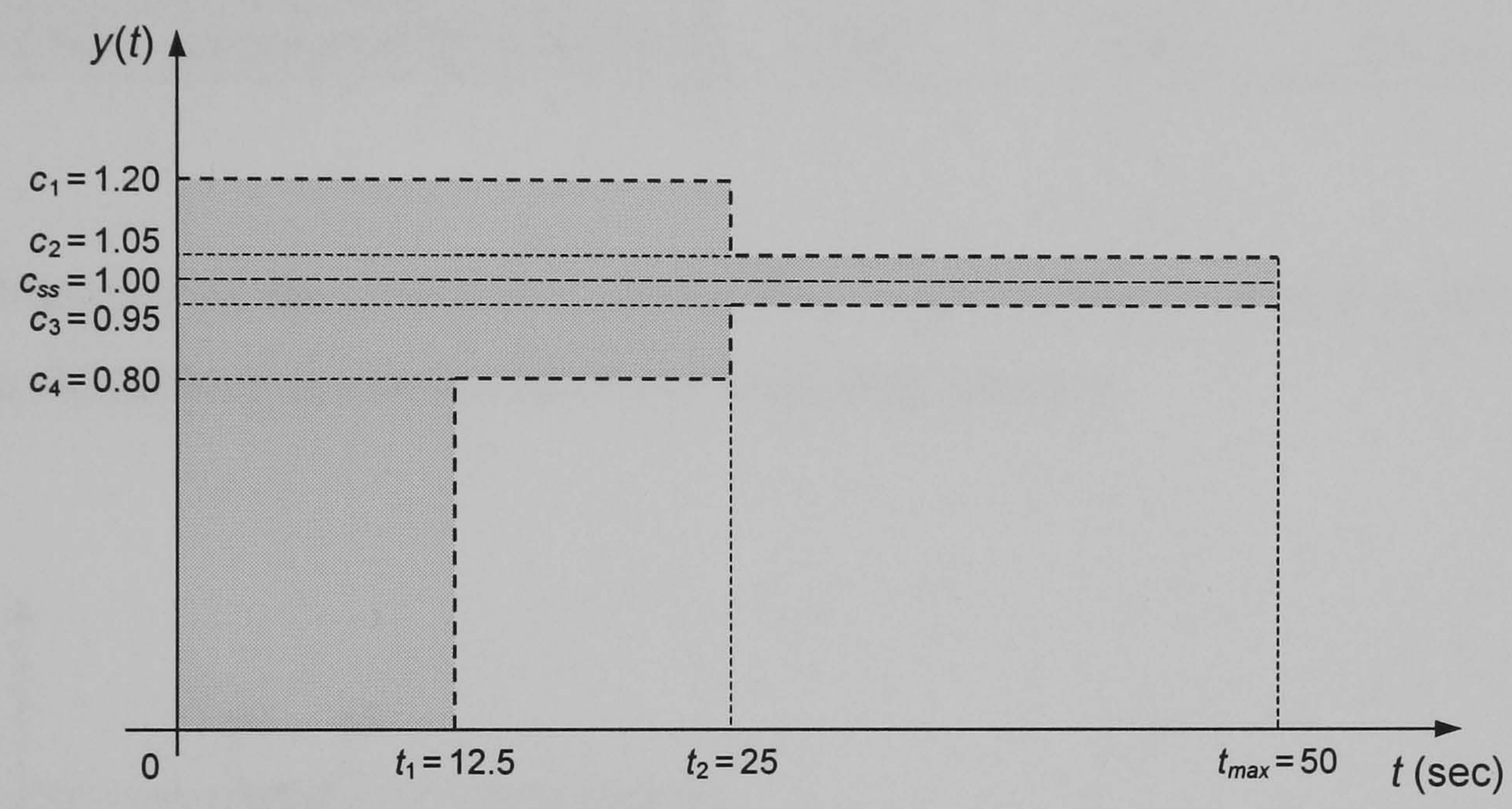


Fig. 3.5 *The performance specifications*

This corresponds to a peak overshoot of at most 20% and a 5% settling time of at most 25 sec. Note that the lower boundary of the shaded region in Fig. 3.5 also imposes some additional specifications, by rejecting responses with points in the rectangle that results by setting $t_1 = 12.5$ and $c_4 = 0.8$. Experiments have shown that including this additional constraint, by setting $t_1 = t_2/2$ and $c_4 = 2c_{ss} - c_1$ (c_1 mirrored around c_{ss}), generally results in qualitatively better responses.

In order to numerically evaluate the objective functions, the closed-loop system was simulated on a computer running the simulation package MATLAB/SIMULINK, using the Runge-Kutta fifth-order numerical integration algorithm with a step size of 0.5 sec and a maximum simulation time, $t_{max} = 50$ sec.

3.4.2 Conventional Objective Functions

The conventional objective functions shown in (3.3) were minimised numerically, using the downhill simplex method of Nelder and Mead (1965). In each case, the minimisation was repeated several times to ensure that the global minimum is obtained. The resulting optimal controller parameters are shown in Table 3.1 below.

TABLE 3.1 Optimal controller parameters using the objective functions in (3.3)

Objective function	K_p	K_i (sec^{-1})	Strictly optimal?
Integrated Absolute Error (IAE)	4.2165	0.3862	Yes (J_S = 0)
Integrated Time and Absolute Error (ITAE)	3.4828	0.3458	Yes (J_S = 0)
Integrated Squared Error (ISE)	5.5658	0.3570	Yes (J_S = 0)
Integrated Time and Squared Error (ITSE)	4.7607	0.3782	Yes (J_S = 0)

The performance of the controllers shown above can be observed in Fig. 3.6, which shows the closed loop system responses to a unit step function.

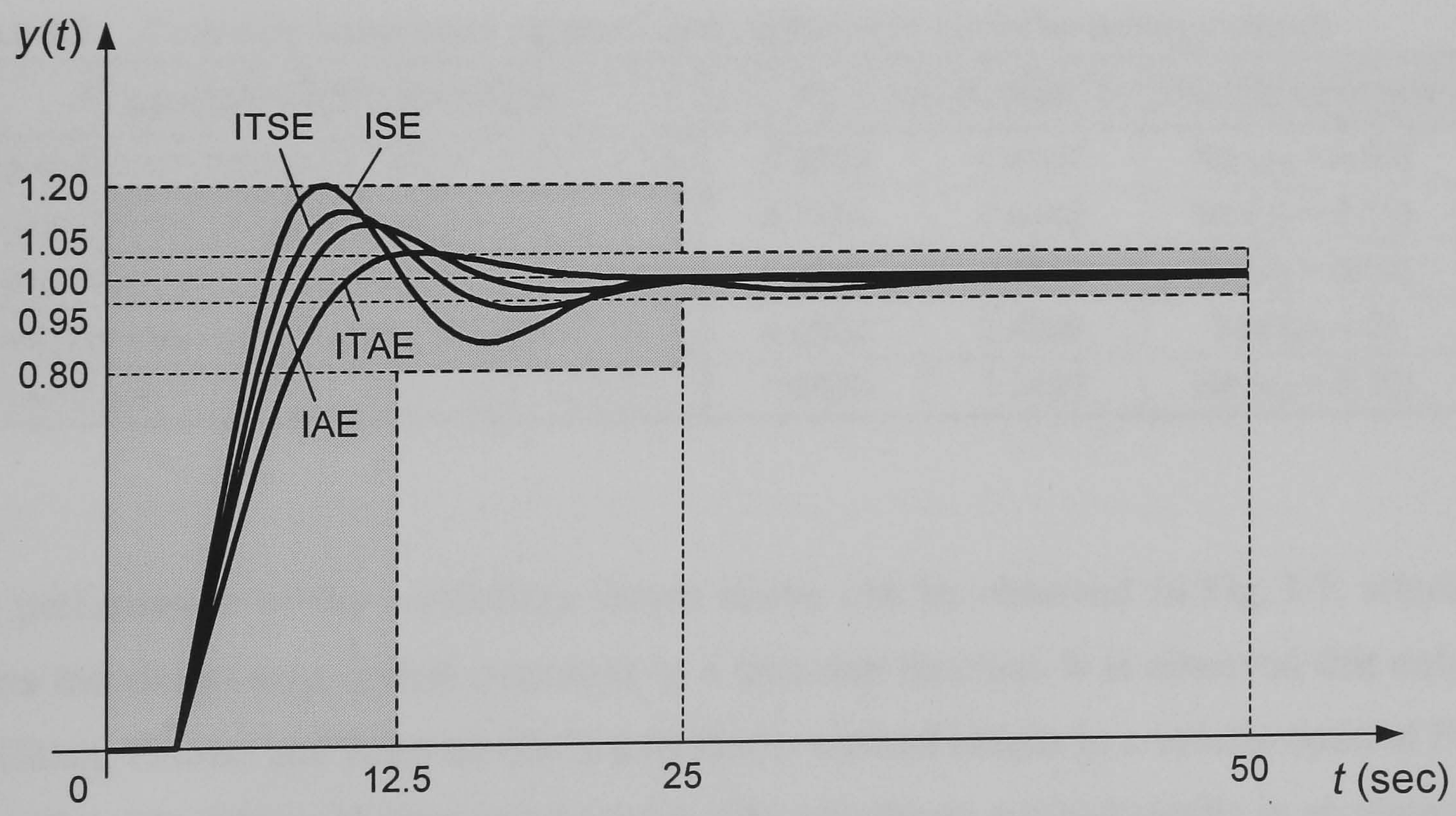


Fig. 3.6 Closed-loop system responses to a unit step function (see Table 3.1)

It is observed that all optimal controllers in Table 3.1 are also strictly optimal (they completely satisfy the specifications in Fig. 3.5). The ISE criterion results in the fastest

response, with its peak value almost at the upper boundary of the specification region, having a peak overshoot of almost 20%. The ITAE criterion results in the slowest and least oscillatory response. Note that the solutions obtained, although strictly optimal, are only four of the infinite number of strictly optimal solutions contained in $\tilde{\mathcal{D}}$. It will be shown later that a much larger number of strictly optimal solutions (theoretically all of them) can be identified using the proposed objective function, J_S.

3.4.3 PI Controller Tuning Methods

The control problem in Fig. 3.4 was also solved using five standard PI controller tuning methods based on the characteristics of the controlled process. These are the classical step and frequency response methods due to Ziegler and Nichols (1942), the method by Chien, Hrones and Reswick (1952) optimised for set point tracking with 0% and 20% overshoot, and finally the method by Cohen and Coon (1953). For more information on these methods, refer to Åström and Hägglund (1995). The resulting optimal controller parameters are shown in Table 3.2 below.

TABLE 3.2 Controller parameters obtained using different PI controller tuning methods

PI controller tuning method	K_p	K_i (sec^{-1})	Strictly optimal?
Ziegler-Nichols (step response)	6.0000	0.6667	No (J_S = 0.85)
Ziegler-Nichols (frequency response)	4.7121	0.5442	No (J_S = 0.19)
Chien, Hrones and Reswick (0% overshoot)	2.3333	0.1944	No (J_S = 0.02)
Chien, Hrones and Reswick (20% overshoot)	4.0000	0.4000	Yes (J_S = 0)
Cohen-Coon	7.6560	1.2497	No (J_S = 8.16)

The performance of the controllers shown above can be observed in Fig. 3.7, which shows the closed loop system responses to a unit step function. It is observed that only the Chien, Hrones and Reswick (20% overshoot) method results in a strictly optimal PI controller. The Chien, Hrones and Reswick (0% overshoot) method results in an almost strictly optimal controller having a very overdamped response with no overshoot. The Ziegler-Nichols step and frequency response methods result in controllers having responses with a relatively large overshoot. Finally, the Cohen-Coon method results in a closed-loop system with very low damping, something that was expected since the Cohen-Coon method has been designed mainly for the rejection of load disturbances.

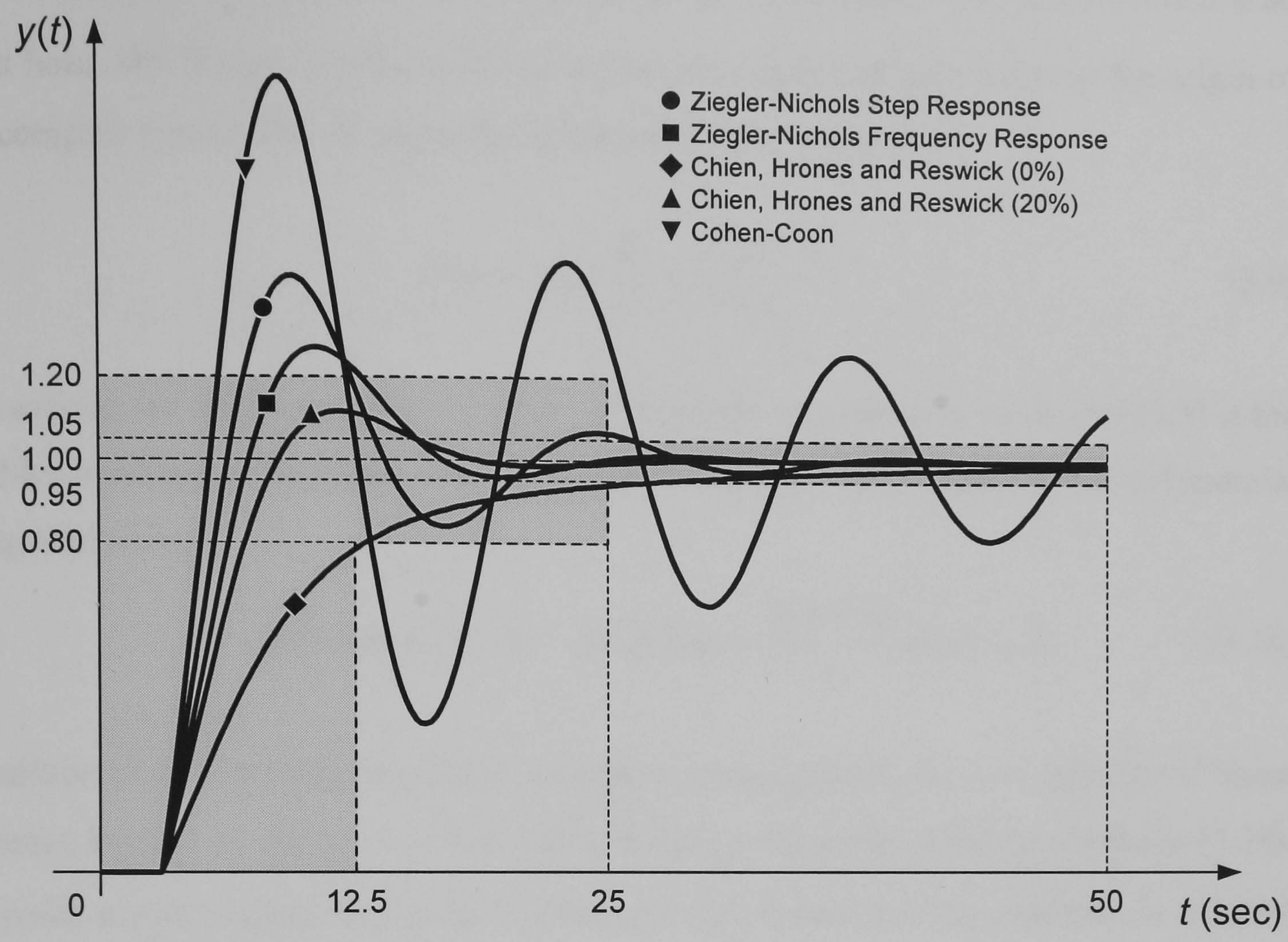


Fig. 3.7 Closed-loop system responses to a unit step function (see Table 3.2)

3.4.4 The Proposed Objective Function

In the preceding analysis it was observed that each of the conventional objective functions and the PI tuning methods results in a single PI controller, which can be thought of as a point in the parameter plane (K_p, K_i). This is not the case with J_S, which results in an *area* in (K_p, K_i) containing all strictly optimal solutions (the set $\tilde{\mathcal{D}}$). In order to better visualise $\tilde{\mathcal{D}}$ and its properties, the parameter plane (K_p, K_i) was augmented with relative stability contours, based on the PID controller design method by Shafiei and Shenton (1997). The procedure is outlined below.

Assume that the process under PI control can be adequately represented by the following standard transfer function.

$$G(s) = \frac{b_0 s^m + b_1 s^{m-1} + \cdots + b_{m-1} s + b_m}{s^n + a_1 s^{n-1} + \cdots + a_{n-1} s + a_n} e^{-sL} \tag{3.8}$$

For the transfer function (3.8) to be realisable, the conditions $n \geq m$, $a_i \in \mathbb{R}$, and $b_i \in \mathbb{R}$, must hold. We further assume that $b_m \neq 0$ (the plant does not have zeros at the origin of the complex plane). The PI controller is assumed to be of the form

$$D(s) = K_p + \frac{K_i}{s} = \frac{K_p s + K_i}{s} \tag{3.9}$$

A condition for *absolute* stability of plant (3.8) when controlled by controller (3.9) is the well-known Nyquist stability criterion, in which the open-loop characteristic polynomial is equated to zero.

$$1 + D(s)G(s) = 0 \quad \text{or} \quad D(s)G(s) = \frac{K_p s + K_i}{s} G(s) = -1 \tag{3.10}$$

Condition (3.10) can be generalised, in order to investigate the *relative* stability of linear systems. Instead of using $-1 + j0$ as the reference point in the stability condition (3.10), the point $a + jb$ is used, with a and b chosen in such a way that the condition is satisfied if and only if the closed-loop system achieves a specific gain or phase margin. The condition for relative stability is shown below.

$$D(s)G(s) = \frac{K_p s + K_i}{s} G(s) = a + jb \tag{3.11}$$

For example, setting $a = -1/GM$ and $b = 0$, the condition is satisfied only for systems having a gain margin *GM*. Similarly, setting $a + jb = e^{-j(\pi - PM)}$, the condition is satisfied only for systems having a phase margin *PM*. To facilitate the algebraic development of the analysis, Eq. (3.11) is rearranged (with $s = j\omega$) as

$$j\omega K_p + K_i = \frac{j\omega(a + jb)}{G(j\omega)} = R(\omega) + jI(\omega) \tag{3.12}$$

where $R(\omega)$ and $I(\omega)$ are the real and imaginary parts of $j\omega(a + jb)/G(j\omega)$, respectively. Equation (3.12) can be used to compute relative stability boundaries in the parameter plane (K_p, K_i). When $0 < \omega < \infty$, the stability boundaries for specific gain/phase margins are governed by the following equations.

$$K_i = R(\omega) \quad \text{and} \quad K_p \omega = I(\omega) \tag{3.13}$$

Quantities $R(\omega)$ and $I(\omega)$ can be easily computed for different values of ω using a computer, resulting in a process-dependent set of gain margin and phase margin contours in the parameter plane (K_p, K_i), as shown in Fig. 3.8 below.

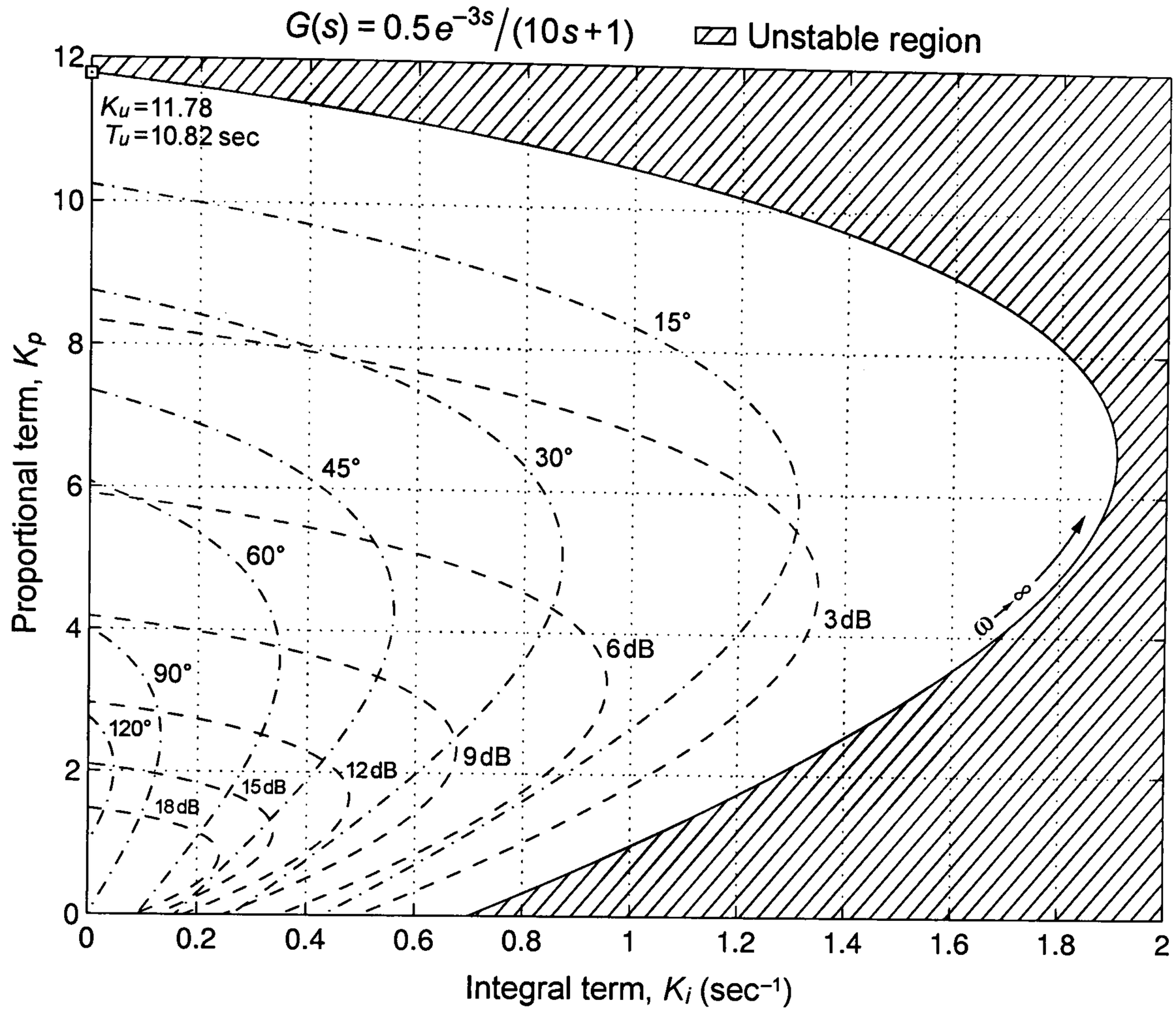


Fig. 3.8 Controller parameter plane including relative stability contours

It is observed from Fig. 3.8 that all stabilising PI controllers (with $K_p \geq 0$, and $K_i \geq 0$) are compactly represented by the white region, making it very easy to visualise the obtained results and their relative stability properties. This is an advantage of the PI controller representation (3.9), and is due to the fact that the controller parameters appear linearly in $D(s)$. This representation also has the advantage that the effect of each term is proportional to the value of the corresponding parameter. Thus, the integral action can be switched off by simply setting $K_i = 0$.

Using the controller parameter plane template of Fig. 3.8, the results of Sections 3.4.2 and 3.4.3, as well as those obtained using the proposed objective function J_S, are shown together in Fig. 3.9, where the boundary of $\tilde{\mathfrak{D}}$ is indicated by the thick line.

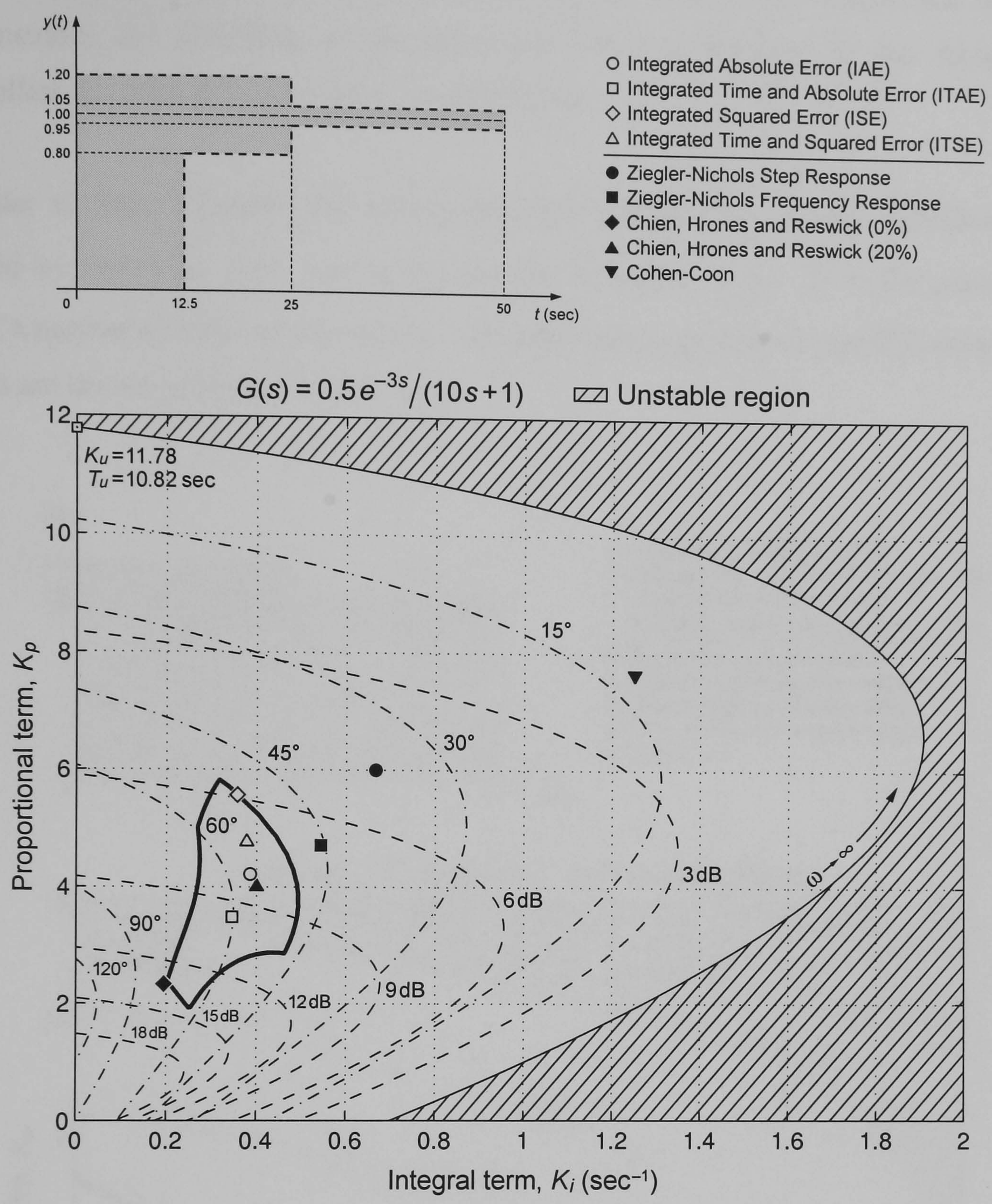


Fig. 3.9 *Controller parameter plane showing the set $\tilde{\mathcal{D}}$ of strictly optimal solutions (bounded by the thick line) for the specifications shown*

It is observed that $\tilde{\mathcal{D}}$ contains solutions with various degrees of relative stability, with gain margins ranging from 5.6 dB to 14.3 dB, and phase margins ranging from 48° to 75°. Note that the shape of the boundary of $\tilde{\mathcal{D}}$ implies that the specifications set for this problem cannot be expressed in terms of gain and phase margin bounds. It is also observed that all solutions obtained by the conventional objective functions are included in $\tilde{\mathcal{D}}$. However, most of the PI tuning methods tested are not contained in $\tilde{\mathcal{D}}$. Of course, this was expected since the optimality criteria used in these methods are not, in

general, associated with the ones shown in Fig. 3.5. The above comparison merely demonstrates the flexibility of the proposed objective function in the design of controllers for *arbitrary* performance specifications.

In order to better illustrate the relationship between the time-domain specifications defined by functions $f_l(t)$ and $f_u(t)$, and the boundary of set $\tilde{\mathcal{D}}$ in the parameter plane, a number of tests were performed using different time-domain specifications. The results are shown in Figures 3.10 to 3.13.

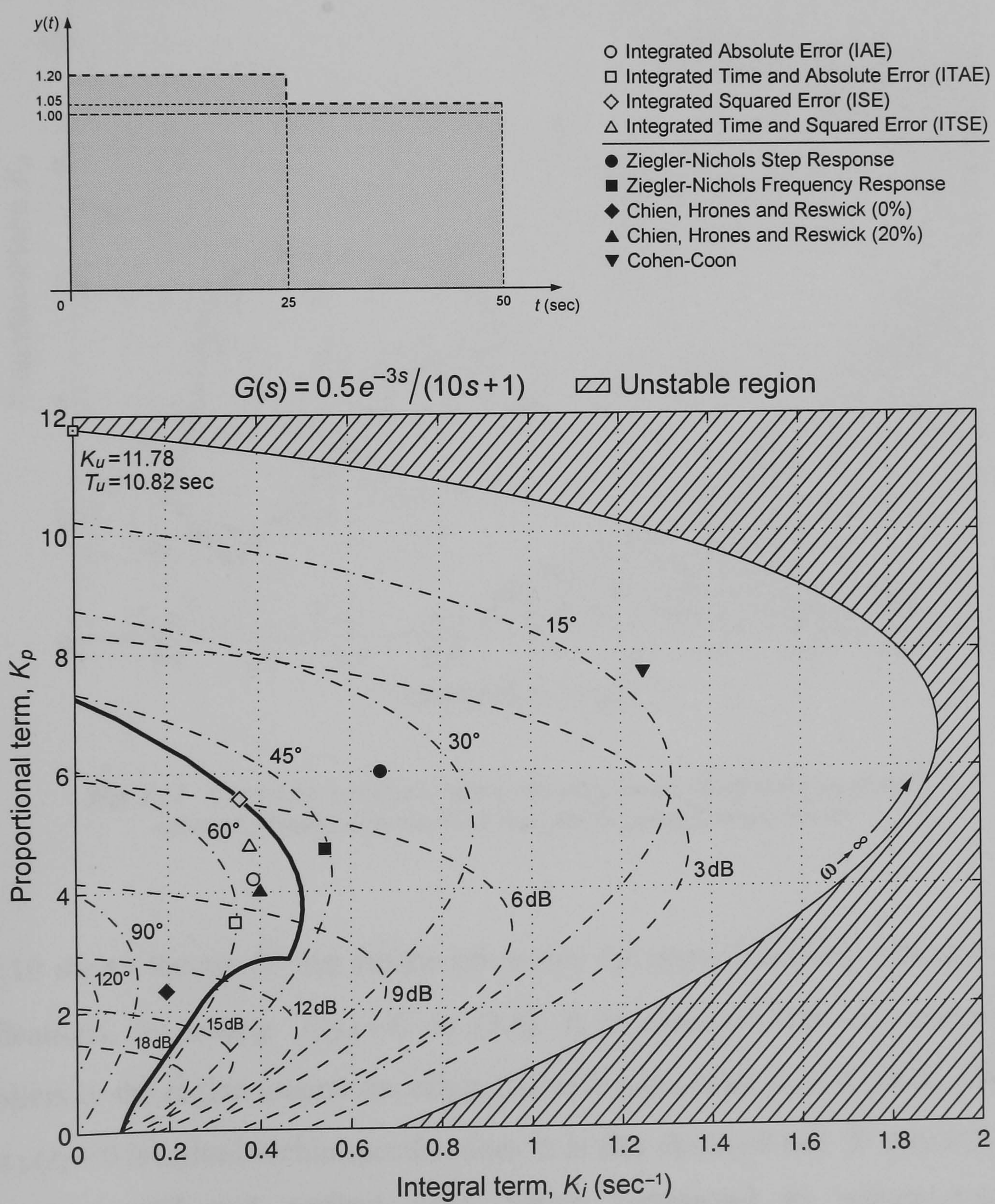
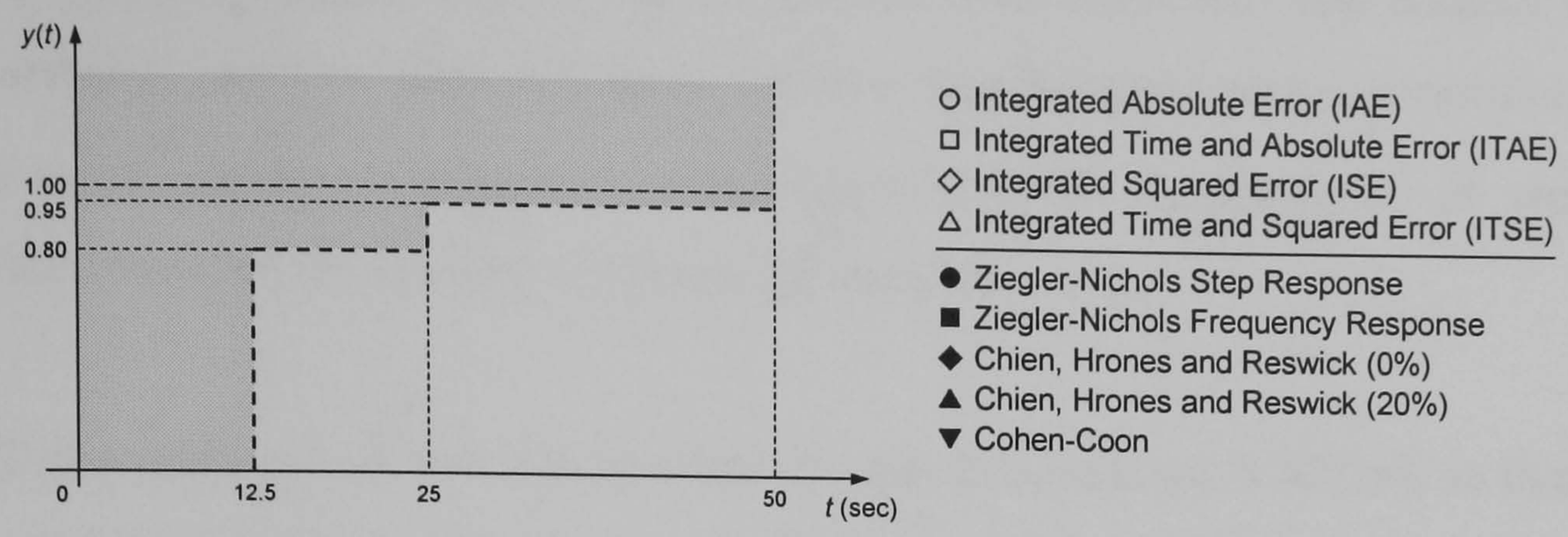


Fig. 3.10 *Controller parameter plane showing the set $\tilde{\mathcal{D}}$ of strictly optimal solutions (bounded by the thick line) for the specifications shown*



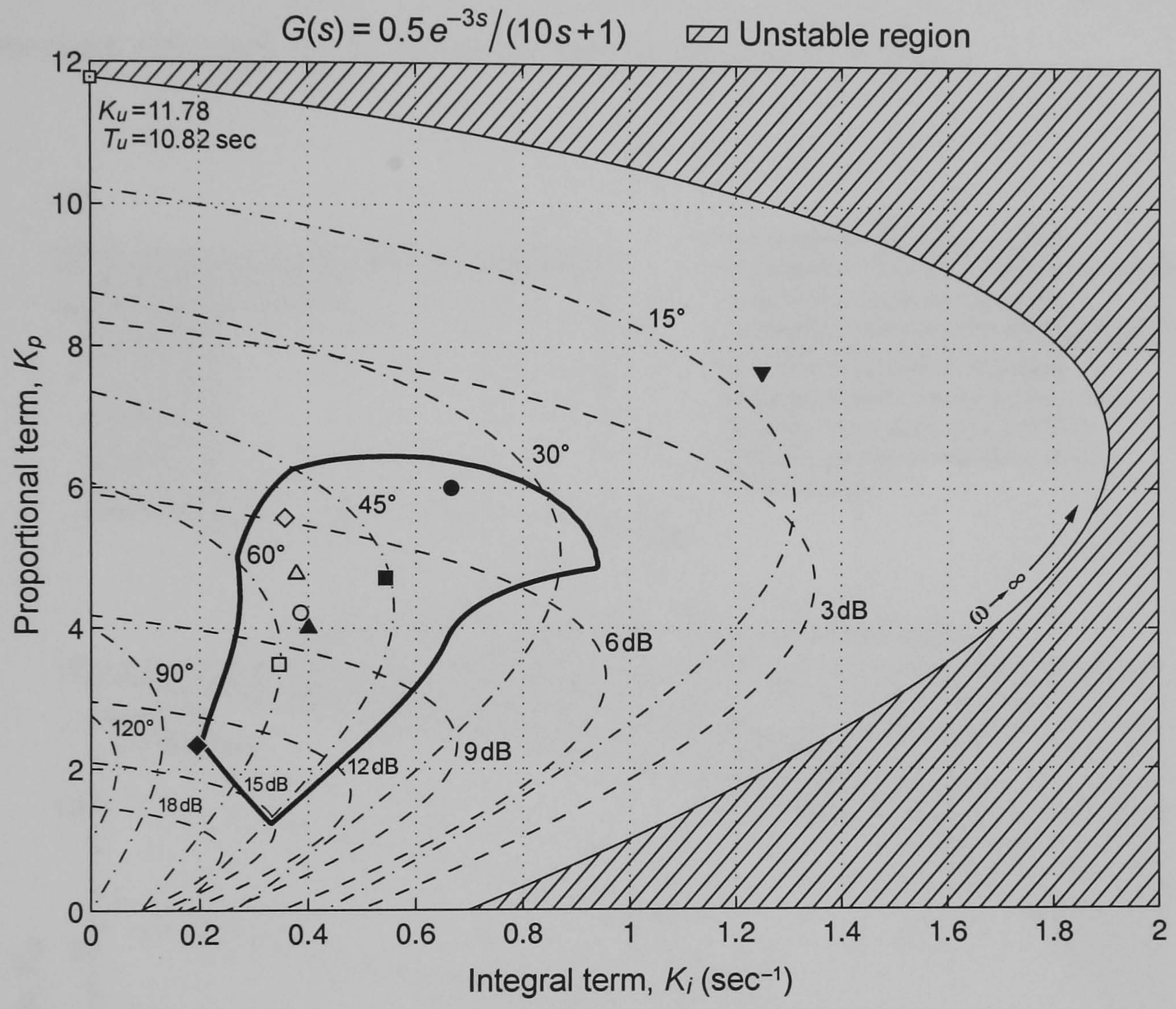


Fig. 3.11 *Controller parameter plane showing the set $\tilde{\mathcal{D}}$ of strictly optimal solutions (bounded by the thick line) for the specifications shown*

Fig. 3.10 shows the set $\tilde{\mathcal{D}}$ that results when only the upper boundary is present in the specifications, by setting $f_l(t) = 0$ in (3.6). It is observed that $\tilde{\mathcal{D}}$ now contains controllers in the region around the origin (including the point $K_p = 0$ and $K_i = 0$, since output $y(t) = 0$ is indeed within specification). It is also observed that $\tilde{\mathcal{D}}$ contains points in the horizontal and vertical axes, which correspond to integral-only and proportional-only controllers respectively. Similarly, Fig. 3.11 shows the results obtained when only the lower boundary is present, by setting $f_u(t) = \infty$ in (3.6). As

expected, $\tilde{\mathcal{D}}$ is now larger than in Fig. 3.9 and contains controllers that achieve lower relative stability margins. This is because the new specifications permit controllers that result in more oscillatory responses with large overshoots. In this case, $\tilde{\mathcal{D}}$ contains most of the solutions obtained by conventional methods.

Fig. 3.12 shows the set $\tilde{\mathcal{D}}$ that results when the specifications are modified, so that they correspond to zero overshoot and a 5% settling time of 25 sec. Since the specifications are now more stringent, set $\tilde{\mathcal{D}}$ is smaller than in the previous cases.

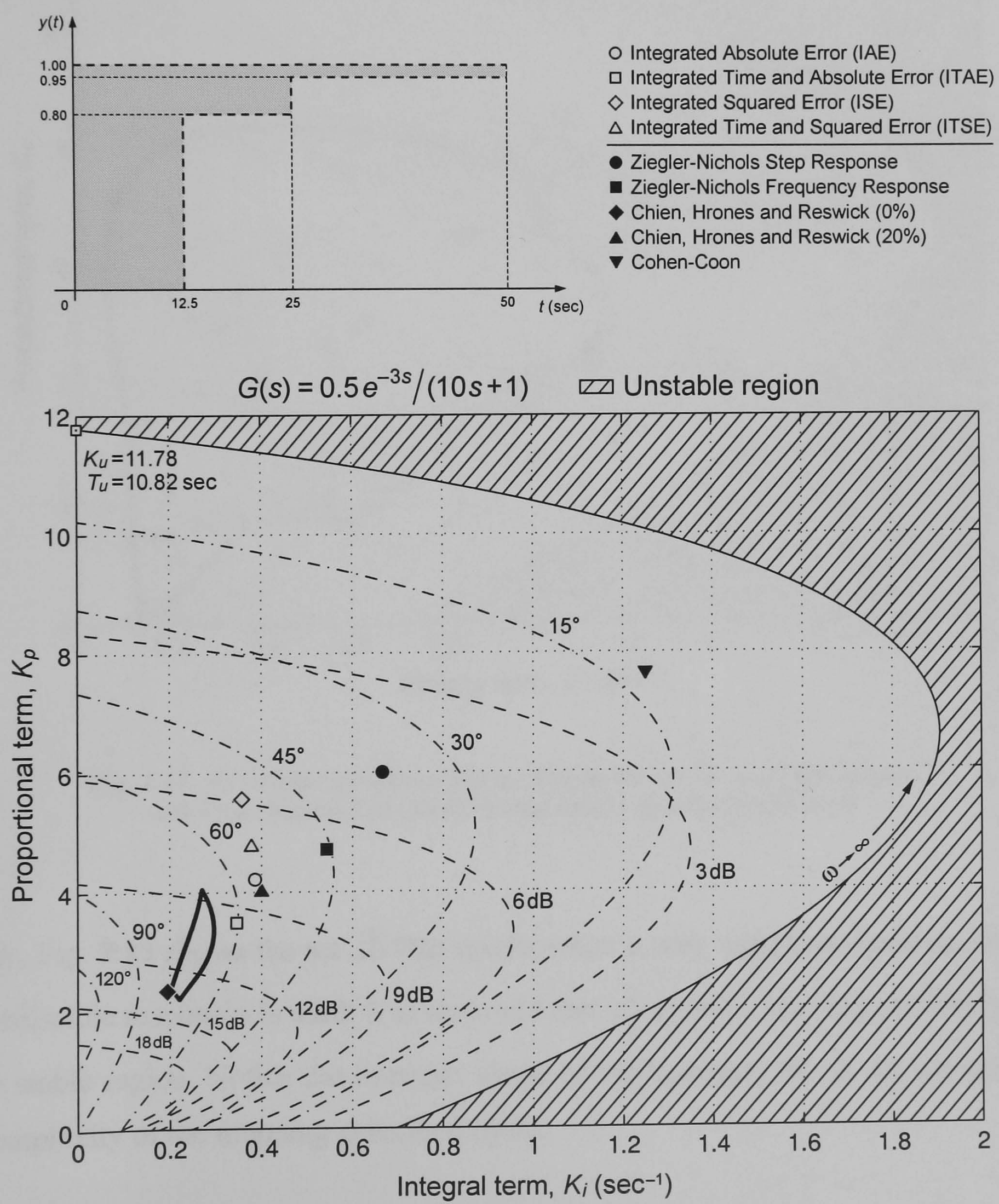
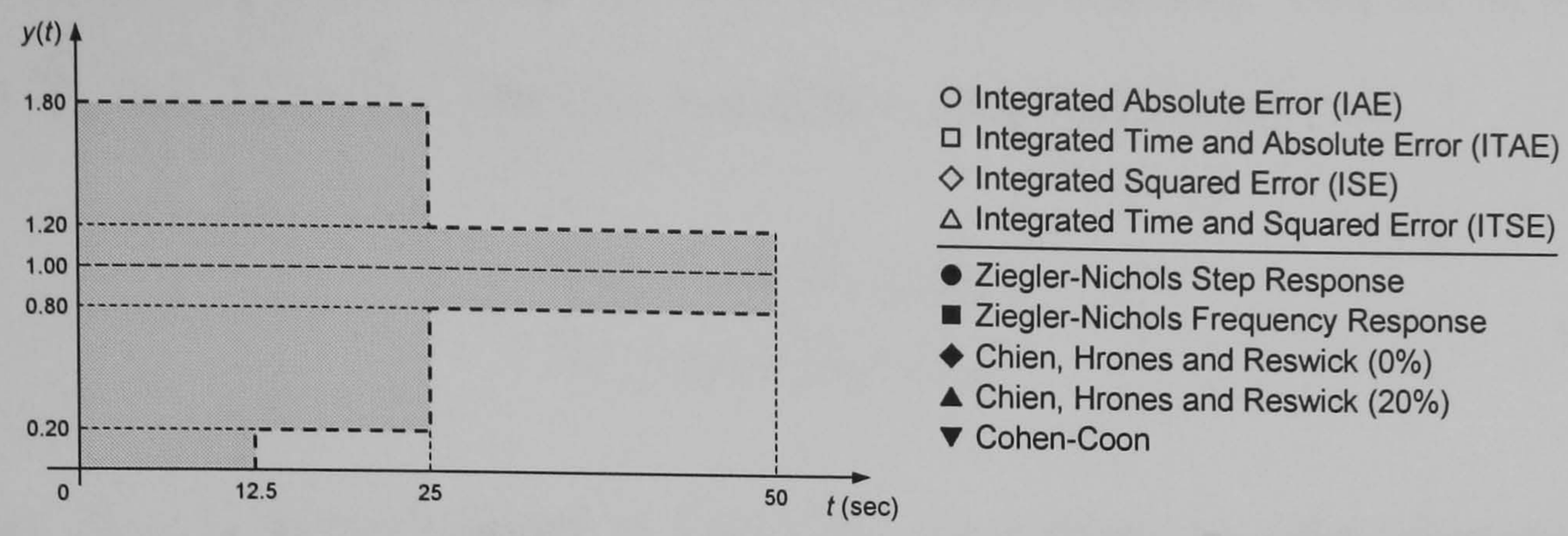


Fig. 3.12 *Controller parameter plane showing the set $\tilde{\mathcal{D}}$ of strictly optimal solutions (bounded by the thick line) for the specifications shown*



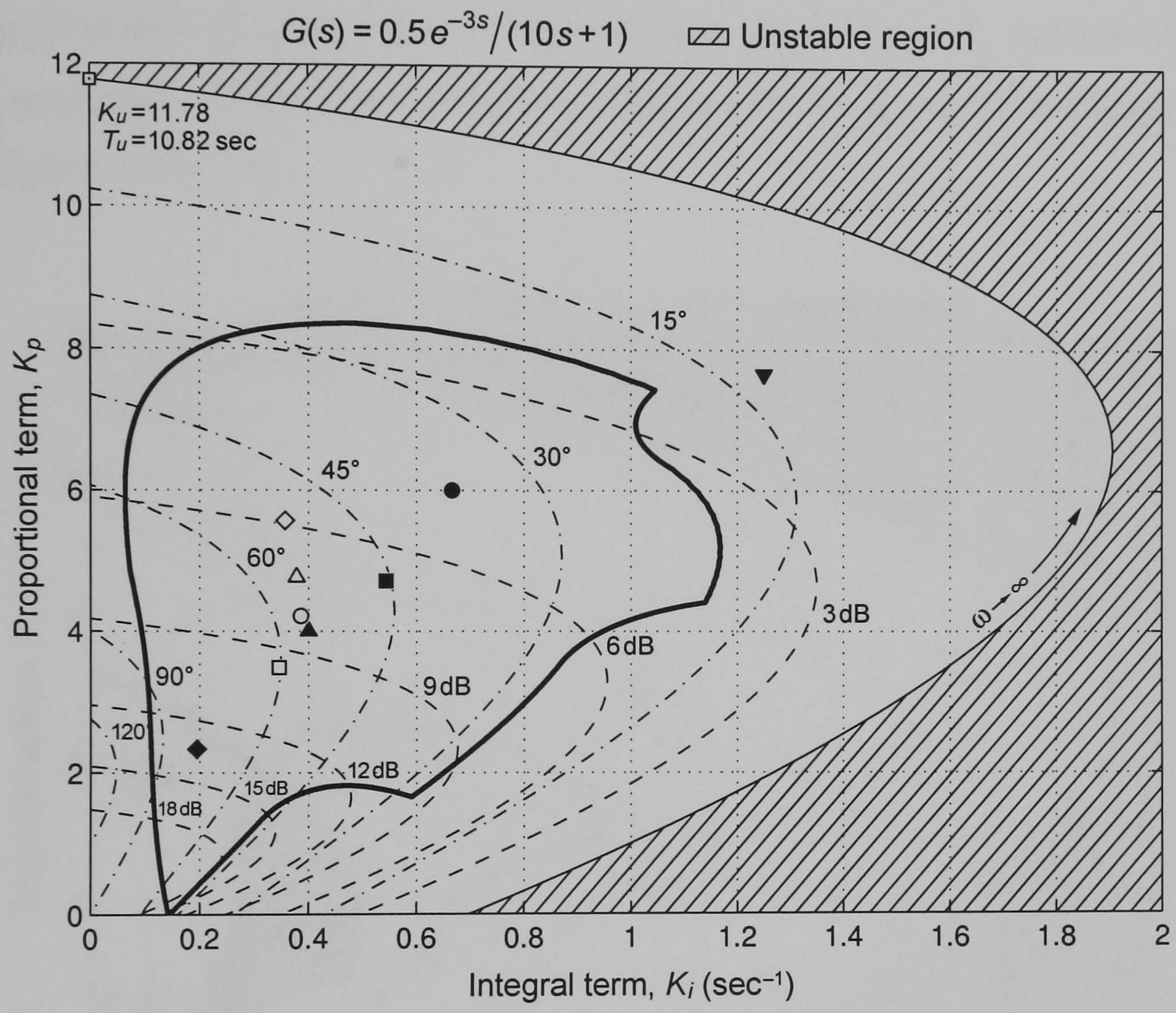


Fig. 3.13 Controller parameter plane showing the set $\tilde{\mathcal{D}}$ of strictly optimal solutions (bounded by the thick line) for the specifications shown

Finally, Fig. 3.13 shows the set $\tilde{\mathcal{D}}$ that results when a very wide time-domain envelope of permissible responses is used. It is observed that $\tilde{\mathcal{D}}$ is now very large, covering most of the stable region. Notice the irregular shape of the boundary of $\tilde{\mathcal{D}}$, an indication of the complexity of the mapping described by J_S.

Let $\mathcal{S}_1$ and $\mathcal{S}_2$ denote the areas in the (y, t) plane, of two arbitrary specifications, both defined in $[0, t_{max}]$ by means of the boundary functions $f_l(t)$ and $f_u(t)$. Also, let $\tilde{\mathcal{D}}_1$

and $\tilde{\mathcal{D}}_2$ denote the corresponding sets of strictly optimal solutions. This can be written as $\mathcal{S}_1 \mapsto \tilde{\mathcal{D}}_1$ and $\mathcal{S}_2 \mapsto \tilde{\mathcal{D}}_2$. Then, the formulation of J_S implies that

$$\begin{aligned} \mathcal{S}_1 \subseteq \mathcal{S}_2 &\Leftrightarrow \tilde{\mathcal{D}}_1 \subseteq \tilde{\mathcal{D}}_2 \\ \mathcal{S}_1 \cap \mathcal{S}_2 &\mapsto \tilde{\mathcal{D}}_1 \cap \tilde{\mathcal{D}}_2 \end{aligned} \tag{3.14}$$

Note that $\mathcal{S}_1 \cup \mathcal{S}_2$ is not mapped to $\tilde{\mathcal{D}}_1 \cup \tilde{\mathcal{D}}_2$, since there can exist responses that entirely fit in area $\mathcal{S}_1 \cup \mathcal{S}_2$, but do not entirely fit in either $\mathcal{S}_1$ or $\mathcal{S}_2$ individually. The above properties are illustrated in the Venn diagram shown in Fig. 3.14. Compare this with Figures 3.9 to 3.13.

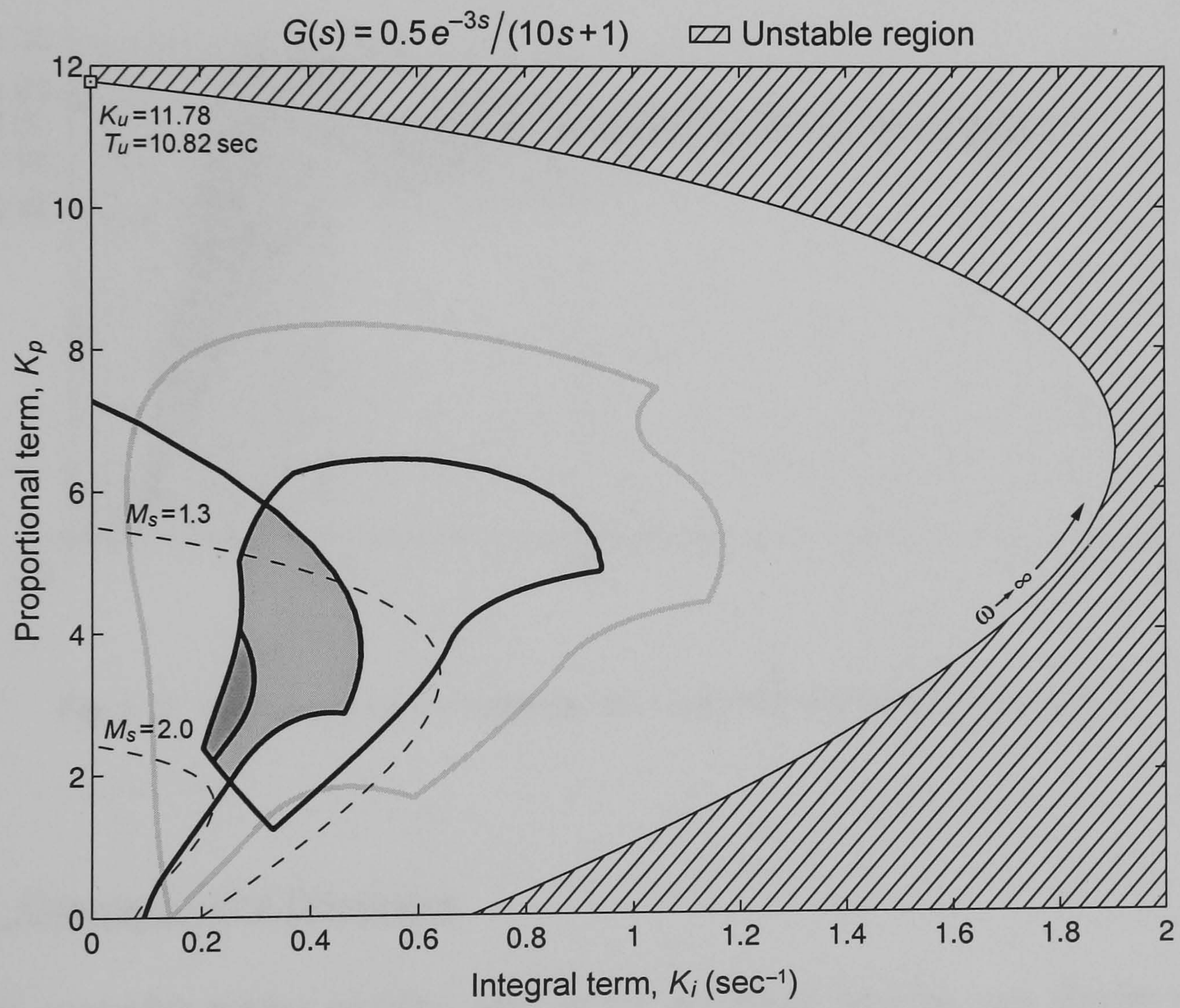


Fig. 3.14 *Venn diagram illustrating properties (3.14)*

Fig. 3.14 also shows the robustness properties of the obtained solutions, in terms of the closed-loop sensitivity to variations in process dynamics, M_s. This is defined as

$$M_s = \max_{0 \le \omega < \infty} \left| \frac{1}{1 + D(j\omega)G(j\omega)} \right| \tag{3.15}$$

Reasonable values for the sensitivity M_s are in the range from 1.3 to 2 (Åström and Hägglund, 1995). It is observed that the majority of controllers in the shaded region are within the reasonable sensitivity range.

In terms of transient responses of the obtained solutions, a very coarse gridding of the stable region (exhaustive search) revealed 24 strictly optimal solutions, uniformly distributed in $\tilde{\mathcal{D}}$, that result in the closed-loop responses shown in Fig. 3.15. Note that the obtained solutions are only a finite subset of the infinite set $\tilde{\mathcal{D}}$.

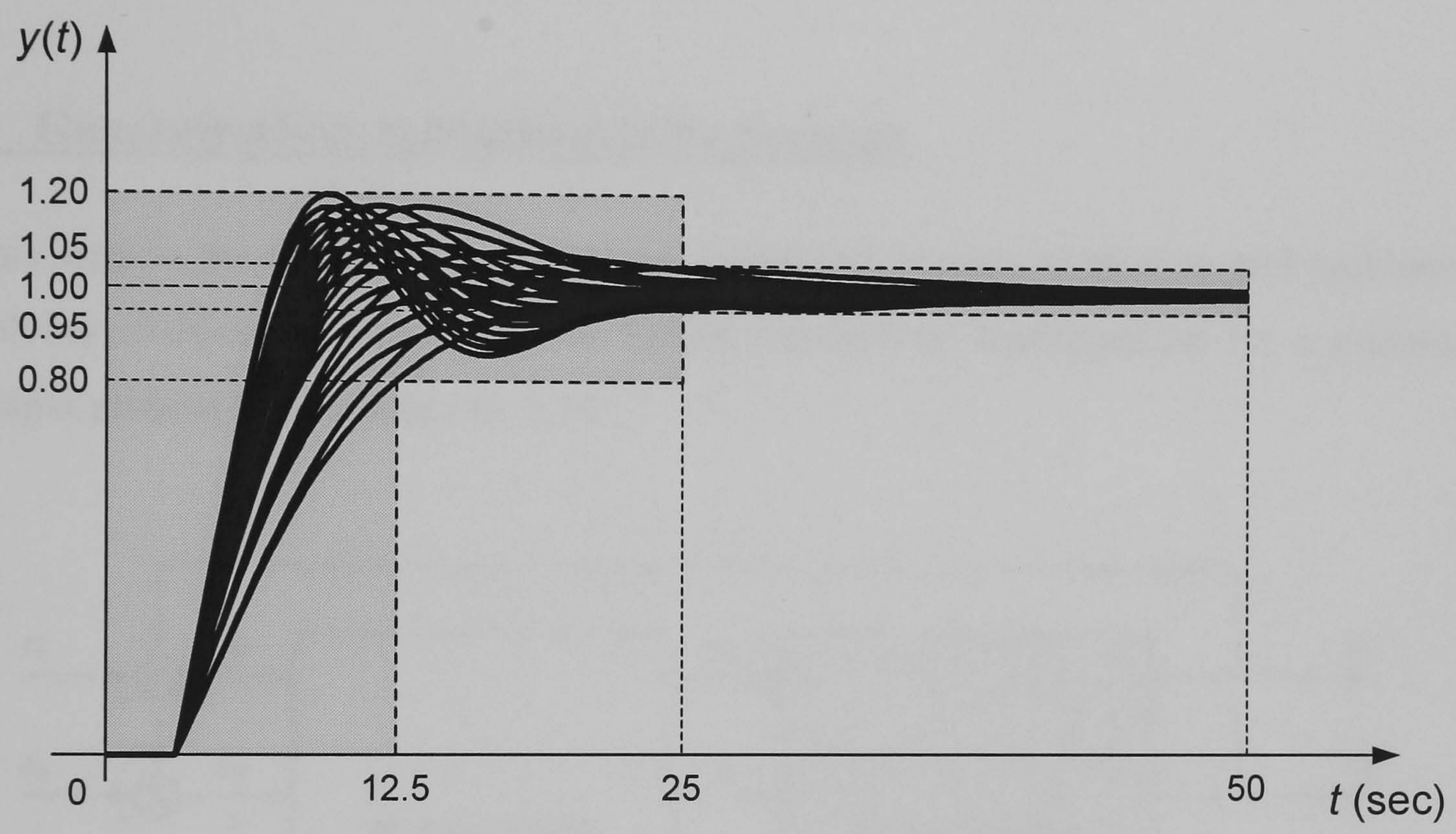


Fig. 3.15 *Closed-loop system responses of 24 uniformly distributed solutions in* $\tilde{\mathcal{D}}$

3.4.5 Comments and Discussion

The PI controller tuning problem used in the preceding analysis was chosen mainly because it results in a two-dimensional search space, thus enabling the visualisation of the obtained solutions. Furthermore, the simplicity of the problem enabled the use of exhaustive search algorithms for the optimisation of the various objective functions, something which guarantees that the solutions obtained are not sub-optimal. Of course, the preceding analysis should not be considered general, since it is only experimental and hence limited to the design examples considered. However, the analysis clearly demonstrates the flexibility and open architecture of the proposed objective function.

The generality of the obtained results will be strengthened in the following chapters, by successfully applying J_S to a number of difficult multivariable control problems.

Recall that GAs search for optimal solutions by evolving a population of *many* candidate solutions in parallel. This makes them especially suited for the minimisation of J_S, which usually results in a *family* of strictly optimal solutions. It will be shown in the following chapters that GAs can successfully identify a finite set of strictly optimal solutions that are *almost uniformly distributed* within $\tilde{\mathfrak{D}}$. This is generally not possible using standard conventional optimisation methods.

3.5 Generalisation to Multivariable Systems

In this section, the objective function J_S is generalised, in order to treat control problems involving multivariable processes. A typical closed-loop configuration for a p-input, q-output process is shown in Fig. 3.16.

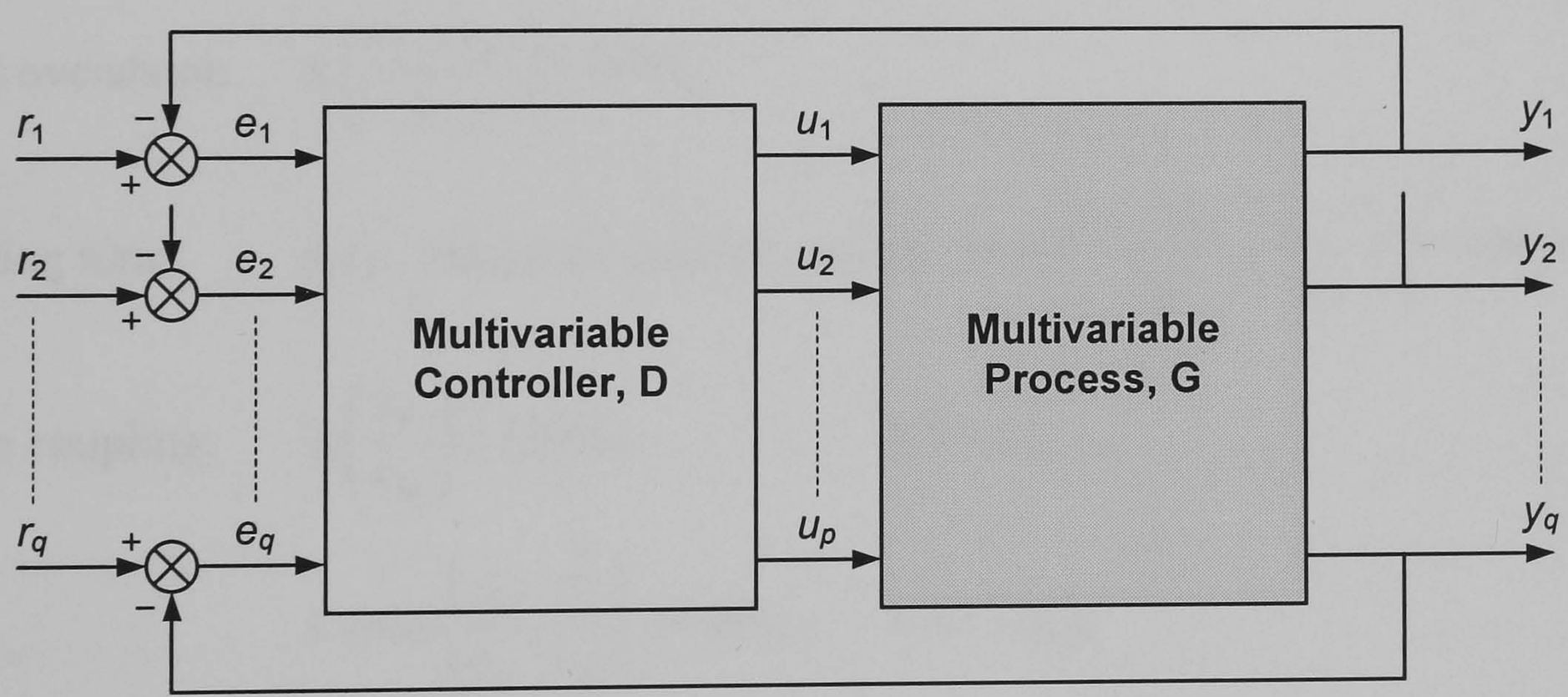


Fig. 3.16 *Typical multivariable closed-loop system*

The objective is to design the q-input, p-output controller so that the q process outputs follow, in some desired way, the corresponding set point signals, and the interactions between the loops are within specification. The set point tracking specifications can be described by the shaded region in Fig. 3.3. The additional loop coupling specifications can be described in a similar way. With reference to Fig. 3.17, the desired response characteristics can be expressed as

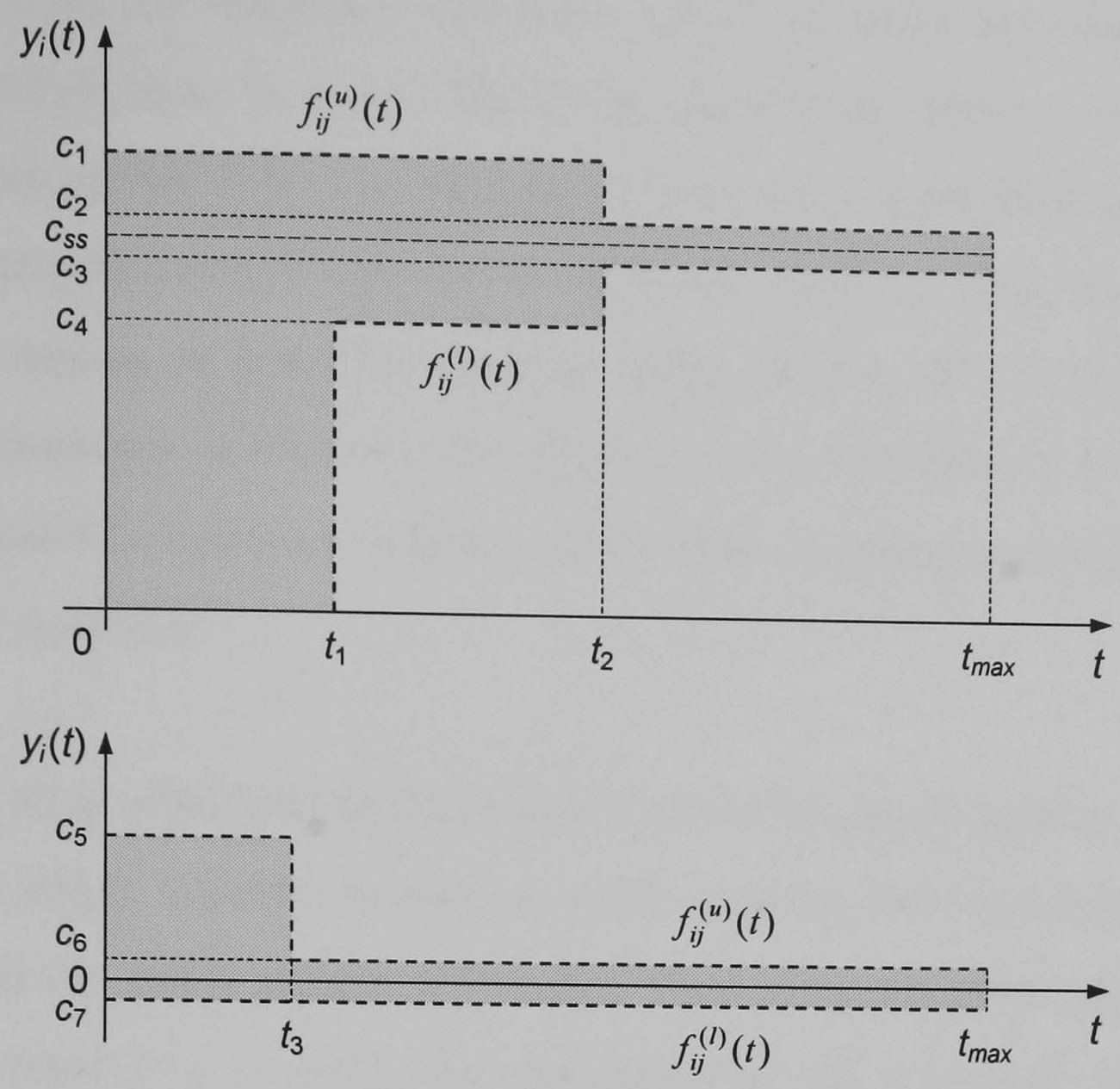


Fig. 3.17 Typical set point tracking (top) and loop coupling (bottom) performance specifications

Peak overshoot: $\leq \left(\dfrac{c_1 - c_{ss}}{c_{ss}}\right) \times 100\%$

Settling time: $\leq t_2$, where the settling time is $\max\left\{\dfrac{c_2 - c_{ss}}{c_{ss}}, \dfrac{c_{ss} - c_3}{c_{ss}}\right\} \times 100\%$

Loop coupling: $\leq \left(\dfrac{c_5}{c_{ss}}\right) \times 100\%$, $\quad 0 \leq t \leq t_3$

$\leq \max\left\{\dfrac{c_6}{c_{ss}}, \dfrac{c_7}{c_{ss}}\right\} \times 100\%$, $\quad t_3 < t \leq t_{max}$

Each output can have different specifications, which also depend on the set point patterns applied to the reference inputs of the closed-loop system. Therefore, in order to evaluate a candidate controller, a number of set point patterns must be applied, and the corresponding sets of specifications must be used to evaluate all q outputs. Let J_{ij} denote the objective function J_S, when used to evaluate output i under set point pattern j, as shown below.

$$J_{ij}(\mathbf{D}) := \int_0^{t_{max}} \left(\max\left\{f_{ij}^{(l)}(t) - y_i(t), 0\right\} + \max\left\{y_i(t) - f_{ij}^{(u)}(t), 0\right\}\right) dt \qquad (3.16)$$

Notice that, in (3.16), the boundary functions can be different for every output and set point pattern combination. In cases where the closed-loop system is square (q-input, q-output) as shown in Fig. 3.16, one possibility is to apply q set point patterns where, in pattern j, a unit step function is applied to reference input r_j, while the remaining $q-1$ reference inputs remain at zero. For each set point pattern, all q outputs are evaluated using objective function (3.16) with the appropriate specifications (set point tracking specifications should be used for output j, while loop coupling specifications should be used for all other outputs).

After evaluating all q objective function elements for set point pattern j, the results can be weighted and added together to form a single number that represents the quality of the controllers for set point pattern j. Finally, when all q set point patterns have been applied to the closed-loop system, the maximum of all resulting objective function elements (indicating the worst performance) can be selected as the final objective function value. The generalised objective function is, therefore, defined as

$$J_M(\mathbf{D}) := \max_{j=1,\ldots,q} \left\{ \sum_{i=1}^{q} w_{ij} J_{ij}(\mathbf{D}) \right\} \tag{3.17}$$

where J_{ij} denotes the objective function for output i under set point pattern j, and w_{ij} denotes the weighting factor of J_{ij}, with $w_{ij} \geq 0$. The higher the weighting factor, the more important the corresponding objective function element becomes. It is observed that q simulations are required for the evaluation of J_M for a single candidate controller. Depending on the complexity of the process and controller, this can result in long execution times and is a disadvantage of the proposed approach. However, this may not be a significant problem in cases where unsupervised search algorithms, such as GAs, are used in the optimisation.

The choice of suitable values for the weighting factors w_{ij} is particularly important when the given performance specifications cannot be completely satisfied (no strictly optimal solutions exist in the search landscape). In such cases, w_{ij} determine the balance of specification violations between the objective function elements J_{ij} that is required to minimise J_M. This means that the location of the optimal point in the search landscape is generally a function of w_{ij}. Conversely, when strictly optimal solutions do exist in the

search landscape, their location is completely unaffected by w_{ij}, because $J_{ij}=0$ for such solutions by the definition of J_{ij}. In such cases, the weighting factors w_{ij} can still affect the convergence of the search algorithm, since their values affect the search landscape. However, the search algorithm should eventually converge to the same strictly optimal solution set, irrespective of the values of w_{ij}, provided that $w_{ij}>0$ for all i and j.

Consider the situation where no strictly optimal solutions exist in the search landscape, but a solution has been found for which *some* elements J_{ij} are exactly zero. This means that the performance specifications for some particular combinations of i and j have been completely satisfied, but not for the remaining combinations. Now assume that a new solution is discovered for which *all* elements J_{ij} are non-zero, but their weighted values result in a value of J_M that is lower than that of the previous solution. Although the new solution results in violations in all combinations of i and j, it will outperform the previous one for which some combinations completely satisfy the specifications. It clearly follows that modifying the specification envelope by means of $f_{ij}^{(u)}(t)$ and $f_{ij}^{(l)}(t)$ for a particular combination of i and j is not even qualitatively equivalent to modifying the corresponding weighting factor w_{ij}. While the former can potentially create strictly optimal solutions in cases where they did not previously exist, the latter merely affects the balance between existing violations. In order to reduce the importance of a particular objective function element, it may thus be more appropriate to widen the corresponding specification envelope by means of $f_{ij}^{(u)}(t)$ and $f_{ij}^{(l)}(t)$, than to simply decrease w_{ij}.

3.6 Summary

This chapter focused on the application of optimisation methods to control engineering problems. It was shown that this can be performed by expressing the performance of the closed-loop system as a function of the controller to be designed or tuned, by means of a single-valued objective function. A number of commonly used objective functions were presented, and it was argued that, although they are convenient to use in an analytical framework, they can often provide poor controller performance indexes. This can affect the quality of the obtained solutions, since the objective function is normally the only source of information used to guide the search towards the optimal solution set. A novel objective function, denoted by $J_S(\cdot)$, was then proposed for single-input, single-output

systems, which enables the designer to explicitly specify the performance specifications associated with a given problem, in terms of time-domain bounds on the closed-loop responses. The formulation of J_S is such that the set of all controllers that completely satisfy the specifications is precisely the kernel of J_S. This set is denoted by $\tilde{\mathfrak{D}}$, and can be the empty set (in cases where the specifications are unrealisable by the controller), or can even be an infinite set, where an infinite number of equivalent controllers exist that completely satisfy the specifications. Objective function J_S was experimentally analysed using a simple PI controller tuning problem. The obtained results were analysed and compared with those obtained using conventional objective functions, as well as using several standard PI controller tuning methods. Finally, a different objective function, denoted by $J_M(\cdot)$, was proposed, which is a generalisation of J_S to multivariable control systems. Both J_S and J_M can be used as a basis for the design and tuning of general, linear or non-linear controllers of arbitrary structures, because they only require the *numerical* solutions of the differential equations associated with the closed-loop system. Provided that a model of the process exists and is relatively accurate, these can easily be obtained using most standard control system simulation packages.

4 Locating Multiple Optimal Solutions Using Genetic Algorithms

4.1 Introduction

In Chapter 3 it was shown that the objective functions J_S and J_M developed in this work have the important characteristic that they result in a family of strictly optimal solutions, namely the set $\tilde{\mathfrak{D}}$. The analytical identification of this set can be a very difficult task, even for trivial problems. This is mainly due to the high complexity of the mappings defined by J_S and J_M. In a numerical optimisation framework, a more realistic goal would be to identify a finite subset of $\tilde{\mathfrak{D}}$, whose elements are *uniformly distributed* within $\tilde{\mathfrak{D}}$. Genetic algorithms appear to be especially suited for this task, since they search by evolving a population of many solutions in parallel. However, the fact that the population size is finite and must be kept relatively small for practical reasons, causes the members of the population of a simple GA to cluster around specific regions in $\tilde{\mathfrak{D}}$, while leaving other regions unexplored. This chapter focuses on ways of overcoming this important limitation of GAs. A new method based on the techniques of *niche formation* and *speciation* is proposed, and it is shown that it enables the GA to evolve a population whose members are almost uniformly distributed within $\tilde{\mathfrak{D}}$. The proposed method is not limited to the optimisation of J_S and J_M, and can be used in many different search landscapes containing multiple optimal solutions.

4.2 Multiple Optimal Solutions and Genetic Drift

In multimodal search landscapes where many local optima may exist, a simple GA is expected to converge to solutions in the neighbourhood of the global optimum. There may be cases, however, where the locations of other optima are also of interest. Most

importantly, there may be cases where the search landscape contains many equivalent optima, all of which correspond to solutions of equal quality. In such cases, the global optimum does not correspond to a unique optimal solution, but to a *set of optimal solutions* that may be finite or infinite. A number of search landscapes with the above properties are shown in Fig. 4.1 below.

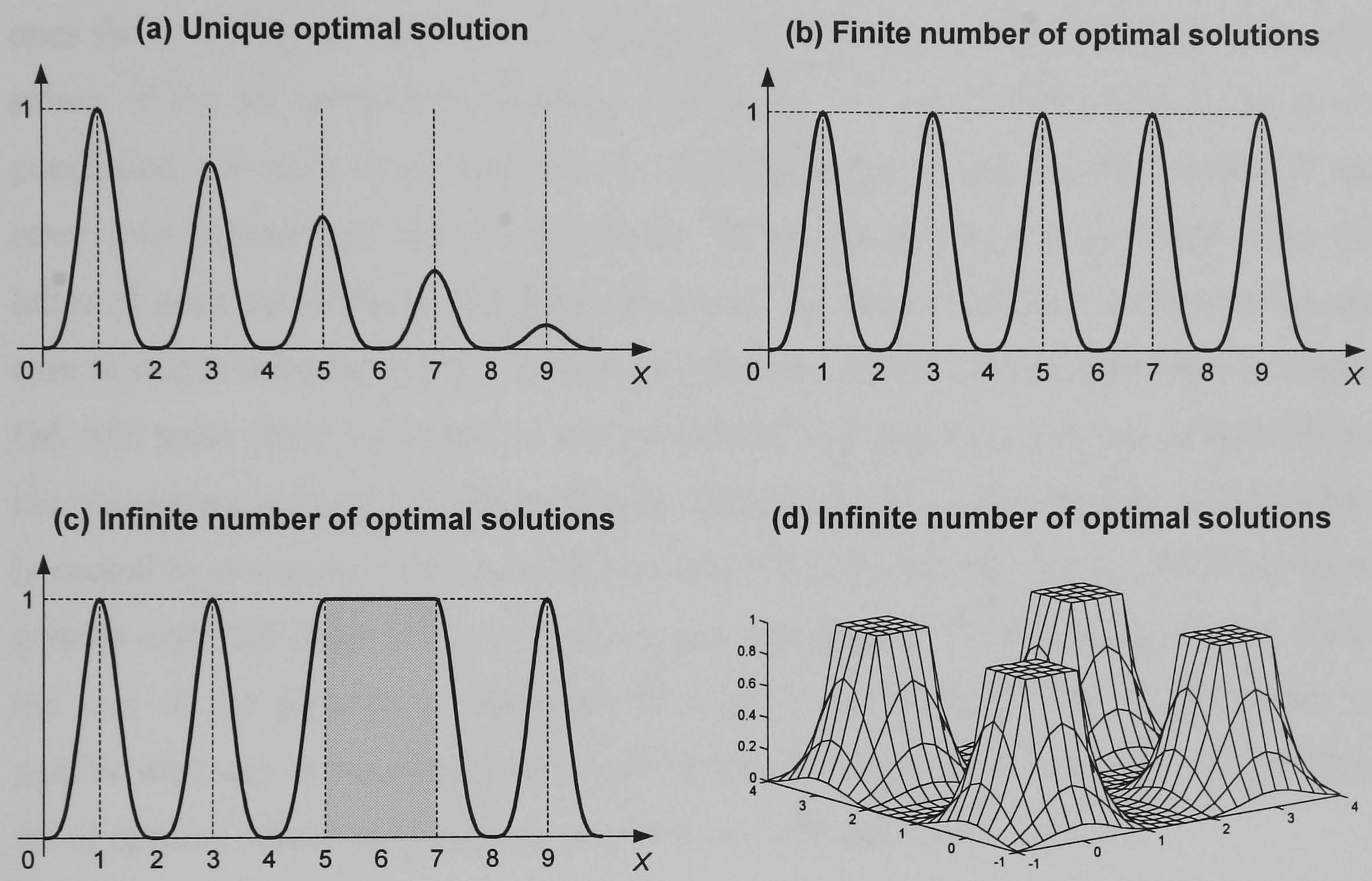


Fig. 4.1 *Examples of multimodal search landscapes*

Assuming a maximisation problem, search landscape (a) contains five maxima which progressively decrease in magnitude. The global optimum is located at the unique optimal solution $x=1$. Although this solution outperforms all others, there may be cases where it is desirable to also find the locations of the remaining four maxima. This is certainly the case in search landscape (b), where all five maxima are of the same magnitude. In this case, the global optimum is achieved by the finite set of optimal solutions $\{1,3,5,7,9\}$. In search landscape (c), the global optimum is achieved by the infinite set of optimal solutions $\{1,3,9\}\cup[5,7]$. This is because for all solutions x in the continuous interval $[5,7]$, the function remains at its maximum value of 1. Similarly, in search landscape (d), the global optimum is achieved by the infinite set of optimal

solutions formed by the union of the four flat regions on the top of the peaks, where the function also remains at its maximum value of 1. Search landscapes (c) and (d) are qualitatively similar to the ones resulting from the minimisation of J_S and J_M, where the global optimum is achieved by all solutions in set $\tilde{\mathfrak{D}}$, and $J_S(\tilde{\mathfrak{D}}) = J_M(\tilde{\mathfrak{D}}) = \{0\}$.

Simple GAs have a particular difficulty in optimising multimodal functions such as the ones shown in Fig. 4.1. Specifically, the population will most likely converge to a small subset of the set of optimal solutions. In the case of search landscape (a), the entire population will most likely converge to the leftmost peak, although the location of the other four optima may also be of interest. Of course, this is not surprising since the leftmost peak outperforms all others. However, the same behaviour is observed in the case of search landscapes (b), (c), and (d), where the optima are all equivalent. A simple GA will most likely converge to only one of the five optima in (b), and will distribute the population unevenly in the continuous optimal regions in (c) and (d). This problem is caused by stochastic errors in sampling due to finite population sizes, and is known as *genetic drift* (De Jong, 1975; Goldberg and Richardson, 1987; Goldberg, 1989a). Since the size of the population cannot be very large for practical reasons, the effects of genetic drift can be significant and must be reduced if an unbiased sample of the entire set of optimal solutions is desired, as in the case of minimising J_S and J_M.

4.3 Niche Formation and Speciation in Genetic Algorithms

In order to overcome the problem caused by genetic drift and maintain appropriate diversity in the population, a number of modifications to the simple GA have been proposed that are loosely based on the natural mechanisms of *niche formation* and *speciation*. In natural ecosystems, living organisms are divided into different species on the basis of their similarities. This enables species to form stable sub-populations which occupy different niches in the environment. In the context of artificial genetic search, niches are analogous to optimal regions in a search landscape, and species are analogous to the members of the population located in these regions. The main objective is to enable the GA to distribute the population evenly among different equivalent niches without bias towards specific niches. A number of proposed schemes incorporating the above ideas into GAs are briefly outlined below.

One of the first attempts to induce niche formation and speciation in genetic algorithms was made by Cavicchio (1970), who introduced a mechanism he called *preselection*. In this scheme, an offspring produced after the crossover of a string pair is required to perform better than the inferior parent in order to assume a place in the population. The inferior parent is then discarded from the population. If the offspring's fitness is lower than that of both its parents, then it is discarded. Furthermore, in order for a mutated individual to assume a place in the population, it must perform better than the individual before mutation. This method helps maintain diversity in the population because strings tend to replace others similar to themselves (one of their parents). This indirectly encourages niche formation and speciation, and helps prevent convergence to a single optimum. Cavicchio claimed to maintain more diverse populations in a number of GA runs with populations of size $N=20$ individuals.

Cavicchio's preselection scheme was later generalised by De Jong (1975) in a scheme he called *crowding*. In this scheme, an offspring is compared with a small number of individuals (typically 2 or 3), randomly chosen from the population. The individual with the highest similarity is replaced by the offspring, where Hamming distance is used as the similarity measure. Like preselection, crowding helps maintain population diversity and encourages niche formation and speciation. A variation of De Jong's crowding was later proposed by Stadnyk (1987), who achieved better results by selecting individuals according to inverse fitness. In this way, offspring tend to replace strings which belong to the same niche and are also inferior to other strings in the niche.

Booker (1982, 1985) proposed a scheme he called *restricted mating*. Restricted mating is based on the observation that species are unlikely to mate with organisms dissimilar to themselves. In this scheme, two individuals in the population are allowed to mate and produce offspring only if they are functionally similar. This facilitates the formation of distinct mating groups (species), which helps promote diversity in the population. The functional similarity between individuals is measured using 'mating templates' – special identifiers in the chromosomes. Only those individuals with matching templates are allowed to mate and produce offspring. The mating templates are not fixed, but evolve along with the rest of the chromosome, adaptively restricting mating between dissimilar species. A similar mating approach is also mentioned in Holland (1987), in the context of classifier systems. Deb and Goldberg (1989) proposed a relatively simple scheme,

called *phenotypic mating restriction*, in which mating is restricted on the basis of similarities between individuals in the decoded parameter space. This has the advantage that it does not depend on the coding scheme used in the chromosome formation. Worth mentioning is the work by Hillis (1992) on *spatially restricted mating*, in which the population evolves on a spatial lattice, and individuals are likely to mate only with those belonging to their spatial neighbourhoods.

4.3.1 Fitness Sharing

Goldberg and Richardson (1987) proposed a practical and effective niche formation and speciation scheme known as *fitness sharing*. In this scheme, individuals that occupy the same niche are forced to share their fitness among each other. Therefore, under fitness sharing, the fitness awarded to an individual is inversely proportional to the number of individuals in the niche it belongs to. Given a number of equivalent niches, this causes underpopulated or unfilled niches to appear more rewarding in comparison with other, overpopulated ones. Eventually, an equilibrium is achieved, where the population is evenly distributed among all equivalent niches. Furthermore, in cases where the niches are not equivalent, the number of individuals allocated to each niche is proportional to the niche's absolute fitness. These properties of fitness sharing have been demonstrated in Goldberg and Richardson (1987), and Deb and Goldberg (1989).

The degree of fitness sharing among individuals is determined by means of a sharing function, $S_f(\Delta_{ij})$, which is a function of the 'distance' Δ_{ij} between strings S_i and S_j in the population. Sharing function S_f determines the degree of membership of strings S_i and S_j to the same niche, based on their similarity. The distance metric $\Delta_{ij} \in \mathbb{R}_+$ can be based on differences in the genotype (such as the strings' Hamming distance), or parameter differences in the phenotype. A typical sharing function returns a value of 1 for identical strings, and its value decreases as the similarity between the two strings decreases. The shared fitness value $\tilde{f}_i$ of the i-th individual in the population is determined by dividing its potential fitness value f_i by the sum of the shares $S_f(\Delta_{ij})$, where $j = 1, 2, \ldots, N$.

$$\tilde{f}_i = \frac{f_i}{\sum_{j=1}^{N} S_f(\Delta_{ij})} \tag{4.1}$$

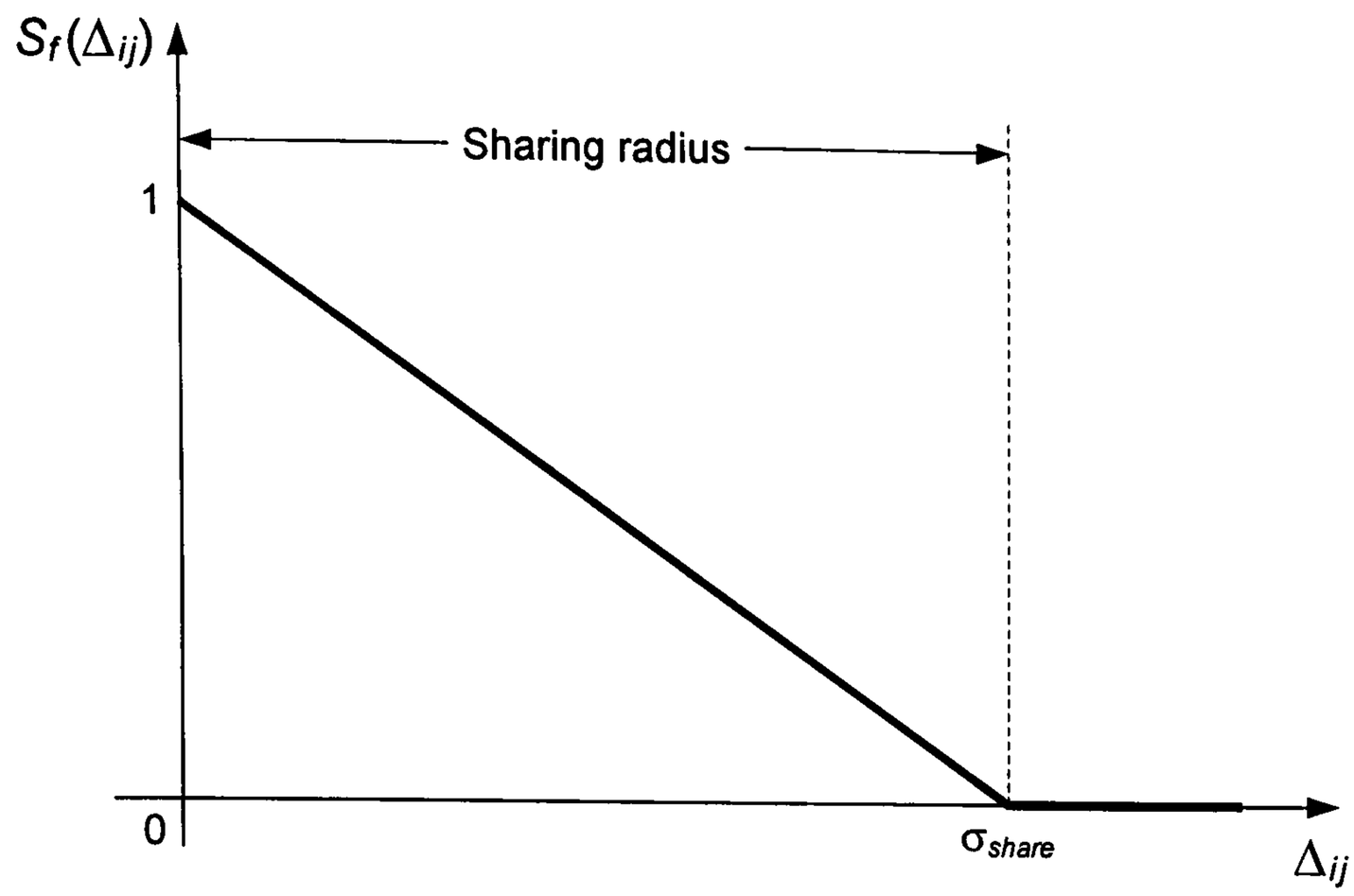


Fig. 4.2 *Triangular sharing function (Goldberg and Richardson, 1987)*

A typical sharing function is shown in Fig. 4.2, where it is assumed that Δ_{ij} is inversely proportional to the similarity of strings S_i and S_j, with $\Delta_{ij}=0$ for identical strings. Parameter σ_{share} determines the 'size' of the niche, and two individuals with $\Delta_{ij} \geq \sigma_{share}$ are considered isolated. It is clear from Eq. (4.1) that when many individuals are in the same neighbourhood, as defined by S_f and Δ_{ij}, their shared fitness values are lower than their potential fitness values, since the shares $S_f(\Delta_{ij})$ are significant. This reduces the reproductive potential of groups of individuals occupying densely populated regions, while isolated individuals retain their potential fitness. As a result, fitness sharing helps maintain diversity in the population by dynamically directing the genetic search towards sparsely populated optimal regions in the search landscape.

A different approach to sharing was later proposed by Beasley, Bull, and Martin (1993) in a method they called *sequential niche*. This method involves multiple GA runs, with each run locating one peak in a multimodal search landscape. After a peak has been located, the search landscape is modified so that the identified peak no longer exists in the landscape. This ensures that the same peak will not be rediscovered. The GA is then restarted and the process is repeated until all peaks have been located. Sequential niche uses a sharing function to suppress the identified peaks from the search landscape. The main difference with fitness sharing is that, instead of the fitness of an individual being reduced because of its proximity to other individuals in the population, the fitness of an individual is reduced because of its proximity to peaks located in previous GA runs.

4.4 The Proposed Niche Induction Method

In this section, a new niche induction method is proposed, which is designed for the optimisation of search landscapes of the form of (c) and (d) in Fig. 4.1. These are qualitatively similar to the ones associated with objective functions J_S and J_M developed in Chapter 3. The required population diversity is achieved by employing an extension of the technique of fitness sharing described earlier, called *adaptive fitness sharing*. Based on theoretical and heuristic justifications, equations are derived for the optimal value of the sharing parameter σ_{share}, which is expressed as a function of the population size, N, the dimensionality of the search landscape, n, as well as certain geometrical properties of the currently identified set of optimal solutions (the set $\tilde{\mathcal{D}}$ in the case of optimising J_S and J_M). This results in a near-uniform sampling of the set of optimal solutions. In addition to that, the population sampling density is adaptively modified in cases where the set of optimal solutions changes in the course of a GA search run. This facilitates the application of the proposed method to on-line optimisation problems.

Note that, in order to improve clarity, $\tilde{\mathcal{D}}$ is thought to contain strictly optimal solutions that may assume a variety of forms. For example, let string S_i correspond to a solution in $\tilde{\mathcal{D}}$. Also, let $\mathbf{x}_i$ denote the parameter vector associated with S_i, and let D_i (or $\mathbf{D}_i$ in the multivariable case) denote the controller associated with $\mathbf{x}_i$. Then, the statements

$$S_i \in \tilde{\mathcal{D}}, \quad \mathbf{x}_i \in \tilde{\mathcal{D}}, \quad D_i \in \tilde{\mathcal{D}} \text{ or } \mathbf{D}_i \in \tilde{\mathcal{D}} \tag{4.2}$$

are all valid, equivalent, and interchangeable. The same applies to J_S, J_M, and any other similar objective function where, for example, the statements

$$J(S_i) = 0, \quad J(\mathbf{x}_i) = 0, \quad J(D_i) = 0 \text{ or } J(\mathbf{D}_i) = 0 \tag{4.3}$$

are all valid, equivalent, and interchangeable. The different equivalent notations shown in (4.2) and (4.3) are all extensively used throughout this work.

4.4.1 Proportionate Fitness Assignment

The reproductive potential of the i-th string in the population of a GA is determined by the string's fitness value f_i, as indicated in Eq. (2.1). Fitness values are non-negative real

numbers assigned to strings, based on their observed performance. Assuming that the optimisation problem involves the minimisation of objective function J, and that $J(S_i)$ denotes the objective function value of string S_i, then the absolute fitness value of S_i is given by the following equation.

$$f_i = \max_{j=1,\ldots,N} J(S_j) - J(S_i) \tag{4.4}$$

This is often called *proportionate fitness assignment* because the reproductive potential of an individual is directly proportional to its observed performance. It is clear that $f_i \geq 0$ and that the lower the value of $J(S_i)$, the higher the value of f_i, with strictly optimal solutions (if they exist in the population) having the maximum fitness value of all N strings. Function J can be any suitable objective function, such as J_S and J_M.

4.4.2 Population Ranking

In proportionate selection, the selection probability p_i of an individual in the population depends on the absolute fitness f_i of that individual, as computed in Eq. (4.4). This can lead to convergence problems, especially in cases where a small number of individuals have fitness values which are significantly higher than those of their competitors. In such cases, these highly fit individuals can often dominate the entire population, while the majority of the remaining individuals are discarded. This can significantly reduce the population diversity, and can also cause the GA to converge prematurely. Furthermore, if parts of the population are characterised by very low fitness variance, proportionate selection may not be able to provide the selection pressure necessary to distinguish between individuals with slightly different fitness values. With finite population sizes, this can lead to individuals with different fitness values contributing the same number of samples in subsequent generations, essentially discarding all available relative fitness information for those individuals.

Population ranking or *rank selection* is an alternative selection method whose purpose is to promote diversity and prevent premature convergence. It achieves this by ranking the individuals in the population according to fitness, and the selection probability p_i of each individual depends on its rank rather than on its absolute fitness. In this approach, all absolute fitness information is totally discarded, while relative fitness information is

preserved. This has the effect of dynamically adapting the selection pressure according to the population fitness variance, since the ratio of expected samples of individuals ranked i and $i+1$ will always be the same whether their absolute fitness differences are high or low.

The population ranking algorithm used in this work was proposed by Baker (1985). In this algorithm, each one of the N individuals in the population is ranked in increasing order of fitness, from 1 to N. Individuals are then assigned new fitness values $\hat{f}_i$, which are given by

$$\hat{f}_i = 2 - \sigma_p + 2(\sigma_p - 1)\frac{i-1}{N-1} \tag{4.5}$$

where it is assumed that the members of the population are ordered according to their absolute fitness, so that S_1 is the least fit individual, S_N is the fittest individual, and S_i is of rank i. Furthermore, it is required that

$$\hat{f}_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{N} \hat{f}_i = N \tag{4.6}$$

It is easy to see that, given the above constraints, the fitness value $\hat{f}_i$ obtained from Eq. (4.5) is equivalent to the expected number of samples s_i of individual i. The amount of selection pressure can be controlled by parameter σ_p, where $1 \leq \sigma_p \leq 2$. When $\sigma_p = 1$ there is no selection pressure, and each individual is expected to contribute precisely one sample to the next generation. When $\sigma_p = 2$ the selection pressure is at its maximum, and S_1 and S_N are expected to contribute precisely 0 and 2 samples, respectively. It is observed that the expected number of samples any one individual contributes to the next generation is bounded in the interval $[2-\sigma_p, \sigma_p]$ and it changes linearly with rank. This fitness assignment strategy has the advantage that it maintains a high fitness variance in the population, and at the same time prevents extremely fit individuals from dominating the population, since $s_i \leq \sigma_p \leq 2$ which means that no single individual is expected to contribute more than 2 samples in the next generation. Note that the average ranked fitness of the entire population always remains constant at unity, irrespective of the choice of σ_p. In cases where there are many individuals with the same absolute fitness values, their ranked fitness $\hat{f}_i$ is averaged over their multiplicity, so that they all have

the same reproductive potential while the global population fitness is kept constant. In Eq. (4.5), fitness values $\hat{f}_i$ change linearly with rank. Exponential and other non-linear functions can also be used, provided that they are strictly monotonic.

4.4.3 Adaptive Fitness Sharing

In conventional fitness sharing methods, such as the one proposed by Goldberg and Richardson (1987) and outlined in Section 4.3.1, parameter σ_{share} is problem-dependent and must be chosen carefully because it directly affects the density of the distribution of the members of the population. Deb and Goldberg (1989) have proposed formulae for computing the optimal value of σ_{share}, assuming that the number of niches is known, and that they are evenly distributed in the search space. However, in most optimisation problems, the validity of these assumptions cannot be guaranteed. The niche size, which directly affects the optimal value of σ_{share}, is rarely known *a priori*. Furthermore, even if σ_{share} is chosen correctly for a given search landscape, the size, shape and location of the niches may change in cases where the search landscape changes during the course of a GA run. In such cases, σ_{share} must be adapted to its new optimal value while the genetic search progresses. In this section, a new fitness sharing method is proposed, in which the optimal value of σ_{share} is estimated as a function of N and n. The optimality of σ_{share} is retained throughout the genetic search, by dynamically adapting the scaling of the search parameters using the information contained in the population at each generation.

Let M denote the number of strictly optimal solutions that are contained in the current population of N individuals. Obviously $M \leq N$ and without loss of generality, assume that the members of the population are ordered in such a way that these M individuals appear first in the population. Therefore, $\{S_1, \ldots, S_M\} \subseteq \tilde{\mathcal{D}}$ and $\{S_{M+1}, \ldots, S_N\} \not\subset \tilde{\mathcal{D}}$. Each solution in the population consists of n parameters. The entire population can, therefore, be expressed as the following $N \times n$ matrix.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \end{bmatrix}^T \tag{4.7}$$

where $\mathbf{x}_i = \begin{bmatrix} x_{i1} & x_{i2} & \cdots & x_{in} \end{bmatrix}^T$ denotes the parameter vector for the i-th solution S_i, and x_{ij} denotes the element in the i-th row and j-th column of $\mathbf{X}$. Now consider $\mathcal{H}$, the n-cuboid

with the smallest n-volume that contains all of the M strictly optimal solutions. This hypercuboid can be thought of as a crude estimate of $\tilde{\mathcal{D}}$, and its size is given by the following vector.

$$\mathbf{h} = \left[\max_i x_{i1} \ \cdots \ \max_i x_{in} \right]^T - \left[\min_i x_{i1} \ \cdots \ \min_i x_{in} \right]^T, \quad i = 1, \ldots, M \tag{4.8}$$

An example of a two-dimensional hypercuboid (a right parallelogram) containing strictly optimal solutions is shown in Fig. 4.3. Set $\tilde{\mathcal{D}}$ is also shown for comparison. This example is taken from the PI controller tuning problem in Section 3.4 of Chapter 3.

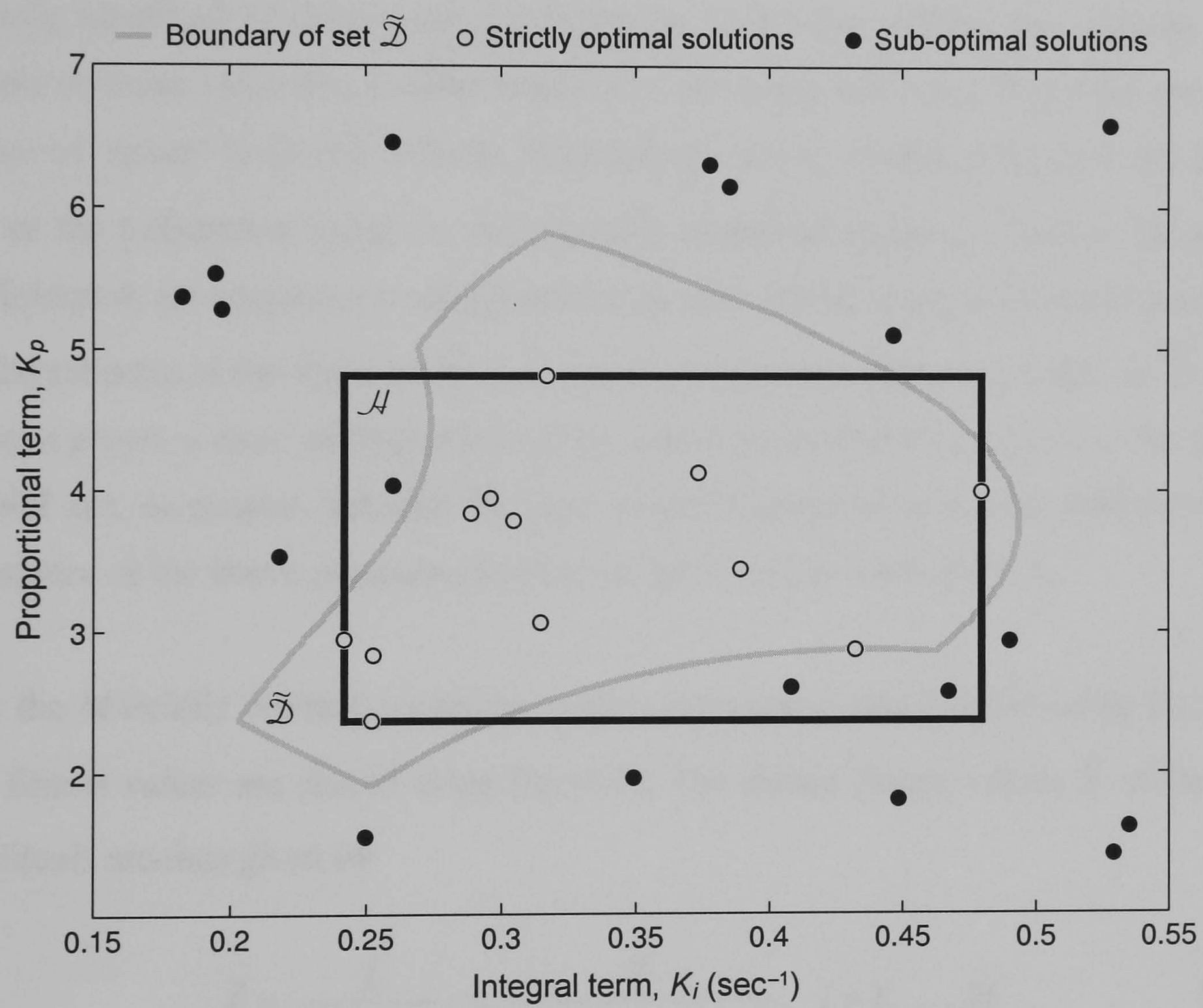


Fig. 4.3 *Example of a 2-cuboid (a right parallelogram) containing all 12 strictly optimal solutions in the current population of 30 individuals*

Hypercuboid $\mathcal{H}$ can expand as new strictly optimal solutions are found, and can also contract in cases where previously identified strictly optimal solutions are no longer strictly optimal due to changes in the search landscape. It will later become apparent that this size adaptation property of the above hypercuboid is directly equivalent to the automatic adaptation of σ_{share} in the proposed adaptive fitness sharing method.

The parameter vectors that correspond to the M strictly optimal solutions are then scaled using vector $\mathbf{h}$, so that hypercuboid $\mathcal{H}$ is transformed into a *unit hypercube*. The scaled parameter vectors are given by

$$\hat{\mathbf{x}}_i = \left[\frac{x_{i1}}{h_1} \; \frac{x_{i2}}{h_2} \; \cdots \; \frac{x_{in}}{h_n} \right]^T, \quad i = 1, \ldots, M \tag{4.9}$$

where h_i is the i-th element of $\mathbf{h}$ and denotes the size of the i-th edge of hypercuboid $\mathcal{H}$, and it is assumed that all elements of $\mathbf{h}$ are non-zero. This parameter vector scaling is necessary in order to obtain an acceptable visualisation of the shape of $\tilde{\mathcal{D}}$, based on the currently identified M strictly optimal solutions. Under this scaling, the samples of $\tilde{\mathcal{D}}$ are now contained in a unit n-cube instead of a n-cuboid, and hence they span the same amount of 'space' in all n directions. Note that the above scaling normalises the aspect ratio of the parameters based on the currently identified shape of $\tilde{\mathcal{D}}$, and the scaling coefficients h_i are adapted at each generation as new strictly optimal solutions are found and the estimate of the shape of $\tilde{\mathcal{D}}$ becomes more accurate. Since the shape of $\tilde{\mathcal{D}}$ is not known *a priori*, a mere normalisation of the search parameter ranges prior to the search run will not, in general, achieve the same result. Experimental results illustrating the importance of the above parameter scaling are presented in Section 4.5.6.

After the M strictly optimal parameter vectors are appropriately scaled using Eq. (4.9), their fitness values are shared using Eq. (4.1). The shared fitness values $\tilde{f}_i$ of these M individuals are thus given by

$$\tilde{f}_i = \frac{f_i}{\sum_{j=1}^{M} S_f(\Delta_{ij})} = \frac{f_i}{\sum_{j=1}^{M} S_f\left(\|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\|_\infty\right)}, \quad i = 1, \ldots, M \tag{4.10}$$

where f_i is the original fitness value of the i-th individual prior to fitness sharing. Any vector p-norm can be used as the distance metric Δ_{ij}. The infinity norm was used in this work, because it facilitates the estimation of the optimal value of σ_{share} that is used in the sharing function S_f (see Fig. 4.2). A method for obtaining an estimate of the optimal value of σ_{share}, based on the infinity norm, is proposed in Section 4.4.5. The choice of

phenotypic sharing is based on the fact that the phenotype of a string is not affected by the type of encoding used in the formation of the string. This avoids any topological distortion introduced by the coding scheme. The advantages of phenotypic sharing have also been reported by Deb and Goldberg (1989), who compared it with genotypic sharing based on Hamming distance information.

From Eq. (4.10) it is observed that fitness sharing is not applied to the entire population, but only to the M members of $\tilde{\mathcal{D}}$, whose shared fitness values may have now become lower than the fitness values of the remaining sub-optimal solutions. In order to ensure that strictly optimal solutions always have higher shared fitness values than sub-optimal ones, the remaining $N-M$ sub-optimal individuals are assigned 'shared' fitness values according to the following equation.

$$\tilde{f}_i = \frac{f_i}{\max\limits_{j=1,\ldots,M} \sigma_j}, \quad i = M+1,\ldots,N \tag{4.11}$$

where

$$\sigma_i = \sum_{j=1}^{M} S_f\left(\|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\|_\infty\right), \quad i = 1,\ldots,M \tag{4.12}$$

In the above equations, σ_i denotes the sum of the shares the i-th strictly optimal solution receives. This quantity can be used as a measure of the population diversity in the phenotypic neighbourhood of string S_i. This is discussed in Section 4.5.1.

In the preceding discussion it was assumed that all elements of $\mathbf{h}$ are non-zero. If there are i zero elements in $\mathbf{h}$, with $i = 1,\ldots,n$, the corresponding hypercuboid $\mathcal{H}$ is said to be rank-deficient, and its n-volume is zero. In such cases, the parameters that correspond to the i dimensions where the zeros occur are excluded from the sharing algorithm by removing the corresponding columns of $\mathbf{X}$ and treating the solutions as having $n-i$ dimensions. If all n dimensions are deficient, such as in cases where there is only one distinct strictly optimal solution in the population, or when $M=0$, then $\mathbf{h}=\mathbf{0}_n$ and no fitness sharing is applied. Rank-deficient hypercuboids may transitionally arise in the beginning of the genetic search when M is small, but they quickly disappear as new distinct strictly optimal solutions are discovered.

The time complexity of the proposed fitness sharing method is $O(M^2)$, which is better than the $O(N^2)$ of conventional fitness sharing since M is initially small and gradually approaches N. Note that the time taken to compute Δ_{ij} for a string pair, using the infinity norm, is usually much smaller than the time taken for an objective function evaluation. Hence, the additional complexity of $O(M^2)$ introduced by sharing is not likely to significantly slow down the genetic search.

4.4.4 Sharing Functions

Various sharing functions can be used with the proposed fitness sharing method. The most common ones are presented in this section. Goldberg and Richardson (1987) and Deb and Goldberg (1989) suggested a class of functions with the following properties.

$$\begin{aligned} &S_f(\Delta_{ij}) \in [0,1], \ \forall \Delta_{ij} \\ &S_f(0) = 1 \\ &\lim_{\Delta_{ij} \to \infty} S_f(\Delta_{ij}) = 0 \end{aligned} \tag{4.13}$$

Many functions satisfy the above properties. Goldberg and Richardson (1987) and Deb and Goldberg (1989) proposed the following power law function.

$$S_f(\Delta_{ij}) = \begin{cases} 1 - \left(\frac{\Delta_{ij}}{\sigma_{share}}\right)^{\alpha}, & \Delta_{ij} < \sigma_{share} \\ 0, & \text{otherwise} \end{cases} \tag{4.14}$$

Parameter α is the power factor, which determines the amount of convexity ($\alpha > 1$) or concavity ($\alpha < 1$) of S_f. The linear, triangular sharing function shown in Fig. 4.2 is obtained by setting $\alpha = 1$. Another sharing function which satisfies properties (4.13) and is based on the exponential function is shown below.

$$S_f(\Delta_{ij}) = \begin{cases} \exp\left(\frac{-\beta\Delta_{ij}}{\sigma_{share}}\right), & \Delta_{ij} < \sigma_{share} \\ 0, & \text{otherwise} \end{cases} \tag{4.15}$$

Sharing function (4.15) is always concave, and the amount of concavity is controlled by parameter β, with $\beta \geq 0$. A higher value of β results in a more concave S_f. Several power law and exponential sharing function graphs are shown in Fig. 4.4 for $\sigma_{share} = 1$.

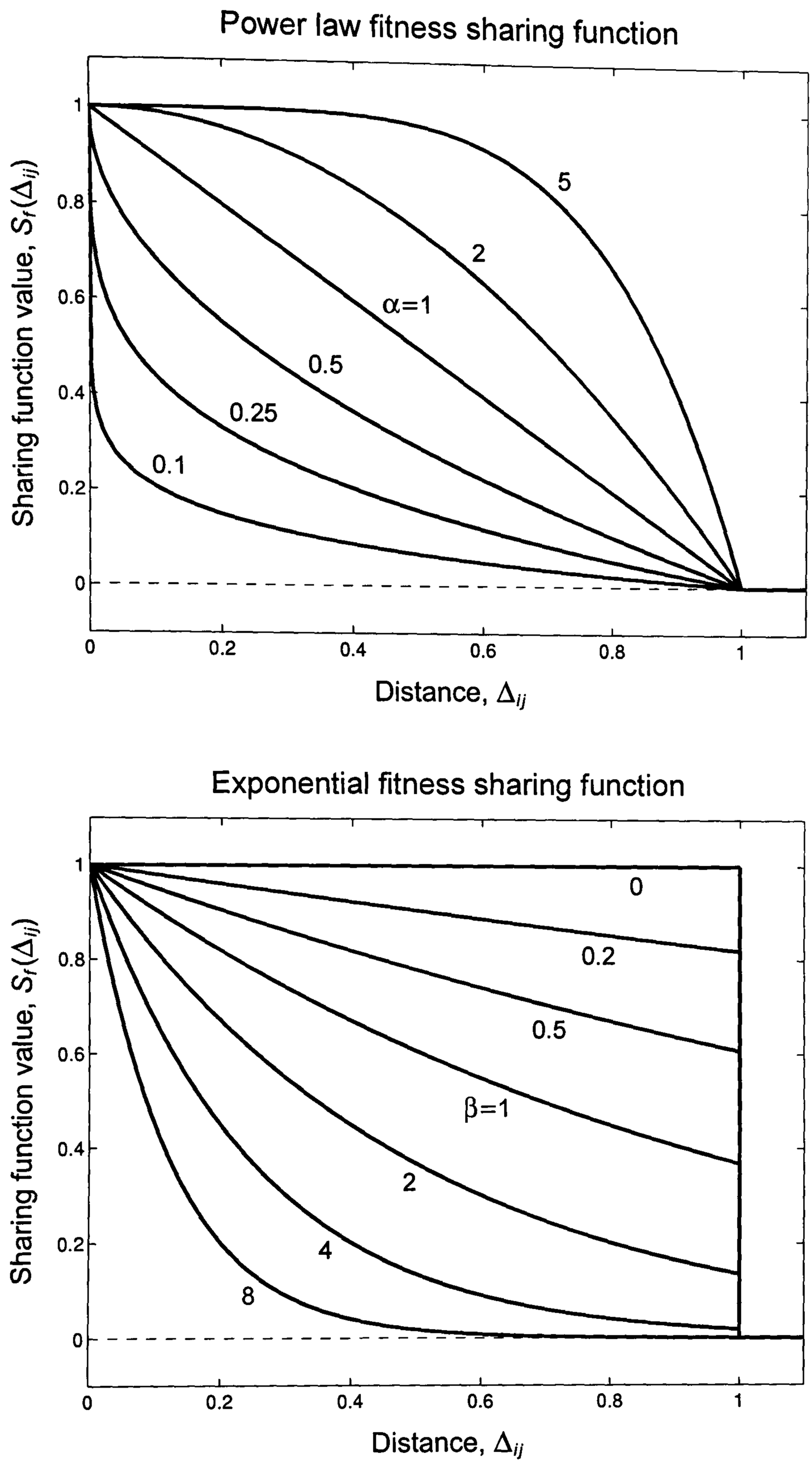


Fig. 4.4 Power law and exponential fitness sharing functions

Using the infinity norm as the distance metric Δ_{ij}, an individual S_i will share its fitness only with those individuals that are contained in the hypercube of edge size $2\sigma_{share}$ and n-volume $2^n\sigma_{share}^n$ which has S_i at its centre. Therefore, fitness sharing between two individuals occurs only when their scaled distance Δ_{ij} is less than σ_{share}.

4.4.5 Optimal Sharing Radius Calculation

It was shown earlier that the proposed fitness sharing method penalises strictly optimal solutions that are in the same neighbourhood, as defined by the sharing function S_f and the sharing radius σ_{share}. This extra pressure on these individuals causes them to move away from each other, since S_f decreases with distance and their shared fitness values will be higher if they are spread out. If σ_{share} is sufficiently small, an equilibrium will eventually be reached, where there will be no two individuals whose scaled distance is less than σ_{share}. If σ_{share} is chosen correctly, this lower bound on the distance between individuals can be maximised, resulting in a near-uniform distribution of the members of the population within $\tilde{\mathcal{D}}$. A method for the calculation of a suitable value for σ_{share} is proposed in this section.

Let $\mathcal{C}$ denote the unit hypercube containing the M scaled strictly optimal solutions. The optimal value of σ_{share} is defined as the maximum value of σ_{share} for which there exists a set of N points within $\mathcal{C}$, such that the distance between any two points is no less than σ_{share}. Note that the above optimality criterion creates enough room in $\mathcal{C}$ for the entire population of N individuals, to allow M to approach N as new solutions in $\tilde{\mathcal{D}}$ are found. Fig. 4.5 shows $\mathcal{C}$ in one, two, and three dimensions. The optimal, uniform distribution of the N solutions, as well as the optimal value of σ_{share} in each case, are also shown.

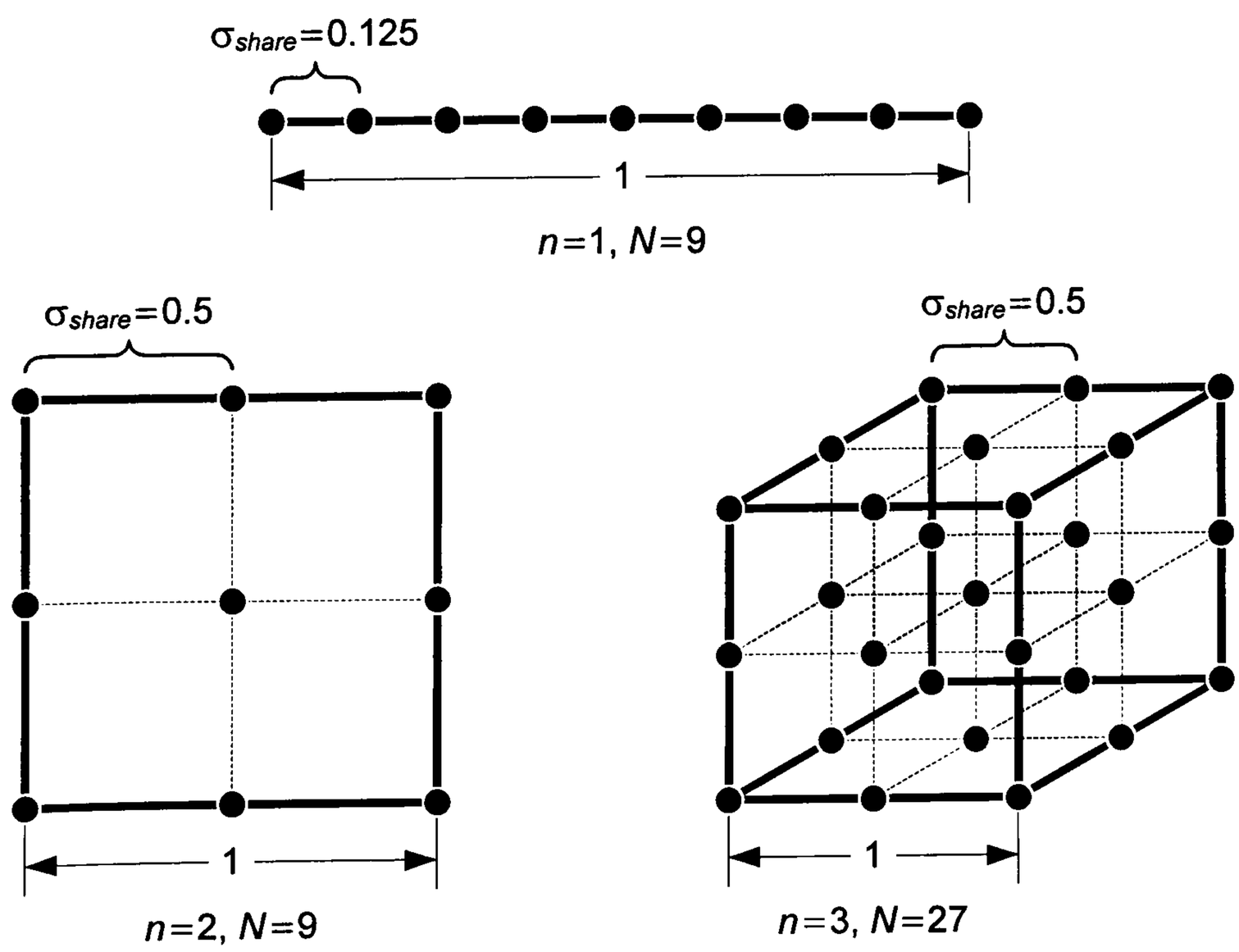


Fig. 4.5 *Unit hypercube $\mathcal{C}$ in one, two, and three dimensions*

Assuming that N is a power of n, it is observed from Fig. 4.5 that each one of the edges of $\mathcal{C}$ contains $\sqrt[n]{N}$ equally spaced points which divide it into $\sqrt[n]{N}-1$ line segments, each of length σ_{share}. The length of each one of the edges of $\mathcal{C}$ is $\sqrt[n]{V_{\mathcal{C}}}$, where $V_{\mathcal{C}}$ denotes the n-volume of $\mathcal{C}$. The optimal value of σ_{share} is thus given by

$$\sigma_{share} \approx \frac{\sqrt[n]{V_{\mathcal{C}}}}{\sqrt[n]{N}-1} \tag{4.16}$$

where the approximate equality becomes an equality when N is a power of n. Let $\tilde{\mathcal{D}}_{\mathbf{h}}$ denote the set $\tilde{\mathcal{D}}$ when transformed using scaling (4.9). The value of σ_{share} obtained from Eq. (4.16) will be optimal only when $\mathcal{C}=\tilde{\mathcal{D}}_{\mathbf{h}}$. In general, however, $\mathcal{C}\neq\tilde{\mathcal{D}}_{\mathbf{h}}$ and hence, the n-volume of $\tilde{\mathcal{D}}_{\mathbf{h}}$ may be different than that of $\mathcal{C}$. A more accurate estimate of the optimal value of σ_{share} may thus be obtained using the following equation.

$$\sigma_{share} \approx \frac{\sqrt[n]{bV_{\mathcal{C}}}}{\sqrt[n]{N}-1} = \frac{\sqrt[n]{b}}{\sqrt[n]{N}-1} \tag{4.17}$$

where b is the n-volume ratio between $\tilde{\mathcal{D}}_{\mathbf{h}}$ and $\mathcal{C}$. The dependency on $V_{\mathcal{C}}$ is dropped, since $\mathcal{C}$ is a unit hypercube and $V_{\mathcal{C}}=1$. Hence, parameter b should be chosen as close as possible to the n-volume of $\tilde{\mathcal{D}}_{\mathbf{h}}$. Recommended values for b are given in Table 4.1 for different values of n, assuming that $\tilde{\mathcal{D}}_{\mathbf{h}}$ is the n-sphere with the largest n-volume that can be placed inside $\mathcal{C}$. Notice that b decreases with increasing n, which indicates that volume mismatches between $\tilde{\mathcal{D}}_{\mathbf{h}}$ and $\mathcal{C}$ can affect the accuracy of (4.17) for large n.

TABLE 4.1 Recommended values for parameter b in Eq. (4.17), for different values of n

n	b	$\sqrt[n]{b}$	n	b	$\sqrt[n]{b}$
1	1	1.0000	6	$\frac{1}{384}\pi^3$	0.6574
2	$\frac{1}{4}\pi$	0.8862	7	$\frac{1}{840}\pi^3$	0.6242
3	$\frac{1}{6}\pi$	0.8060	8	$\frac{1}{6144}\pi^4$	0.5957
4	$\frac{1}{32}\pi^2$	0.7452	9	$\frac{1}{15120}\pi^4$	0.5709
5	$\frac{1}{60}\pi^2$	0.6970	10	$\frac{1}{122880}\pi^5$	0.5491

Note that the recommended values for b shown in Table 4.1 should not be treated as universally optimal, as they are based on the assumption that $\tilde{\mathfrak{D}}_{\mathbf{h}}$ is spherical in shape.

4.5 Experimental Analysis

In this section, the proposed fitness sharing method described earlier is experimentally analysed, in order to demonstrate its effectiveness and illustrate its adaptive properties. The experimental setup consists of the following closed-loop system.

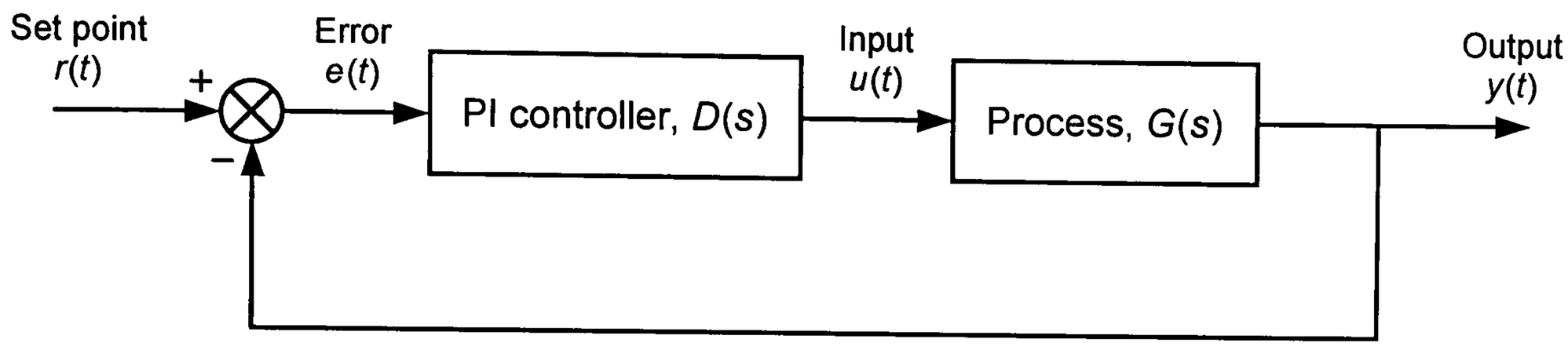


Fig. 4.6 *Experimental setup for the analysis of adaptive fitness sharing*

This system is based on the experimental setup of Section 3.4 in Chapter 3. The transfer functions of the process and controller have the following structure.

$$G(s) = \frac{K}{Ts+1}e^{-sL} \quad \text{and} \quad D(s) = K_p + \frac{K_i}{s} \tag{4.18}$$

The process parameters were set to be $K=0.5$, $T=10$ sec, and $L=3$ sec, but K was allowed to change in certain cases, in order to demonstrate the adaptive properties of the proposed fitness sharing method. The controller performance specifications were the ones shown in Fig. 3.5, namely a peak overshoot of at most 20% and a 5% settling time of at most 25 sec.

A genetic algorithm was used to find optimal values for the controller parameters K_p and K_i, with a generation gap $g=0.9$, reproduction with stochastic universal sampling (Baker, 1987), single-point crossover, and fitness-based reinsertion to implement an elitist strategy. The size of the population was chosen to be $N=80$, in accordance with experimental studies of Grefenstette (1986). The initial, randomly selected population

was left to evolve for a maximum of 100 generations. The two tuning parameters were encoded using the binary alphabet and Gray coding (Caruana and Schaffer, 1988). The string resolution for each of the two parameters was set to 12 bits, resulting in a total string length of $l=2\times 12=24$ bits. The crossover and mutation probabilities were chosen to be $p_c=0.45$ and $p_m=0.01$, respectively, using the guidelines of Grefenstette (1986). In certain cases, however, the mutation operator was disabled by setting $p_m=0$. This enables the assessment of the true capabilities of the proposed fitness sharing method, since no arbitrary diversity is introduced through mutation. If appropriate diversity can be maintained without mutation, similar or better performance should be expected when mutation is present (Goldberg and Richardson, 1987). The proportional terms K_p were assumed to take values from the interval $[0,12]$, and the integral terms K_i from the interval $[0,2]$ sec^{-1}.

The parameters for fitness sharing were set as follows. Function (4.14) was used as the sharing function S_f in Eq. (4.10), with a power factor $\alpha=1$, which is equivalent to the triangular sharing function used in Goldberg and Richardson (1987). The value of the sharing radius σ_{share} was computed using Eq. (4.17), as shown below.

$$\sigma_{share} \approx \frac{\sqrt[n]{b}}{\sqrt[n]{N}-1} = \frac{\sqrt{\frac{1}{4}\pi}}{\sqrt{80}-1} = 0.1116 \tag{4.19}$$

where $n=2$, and b is taken from Table 4.1. Whenever population ranking was applied, the selection pressure in Eq. (4.5) was chosen to be at its maximum value of $\sigma_p=2$.

4.5.1 Population Diversity Measure

In order to quantify the performance of the proposed fitness sharing method, a measure of population diversity is proposed in this section. It is based on the sum of shares σ_i each individual receives, computed using Eq. (4.12). Formally, the degree of diversity or uniformity of a given set of M samples of $\tilde{\mathcal{D}}$ that are contained in $\mathcal{H}$ is defined as

$$U := 1-\frac{\sum_{i=1}^{M}(\sigma_i-1)}{M(M-1)} = 1-\frac{\sum_{i=1}^{M}\sum_{\substack{j=1\\ j\neq i}}^{M} S_f\left(\|\hat{\mathbf{x}}_i-\hat{\mathbf{x}}_j\|_\infty\right)}{M(M-1)} \tag{4.20}$$

It is easy to see that U is always bounded in the closed interval [0,1] provided that S_f satisfies properties (4.13). Assuming σ_{share} is at its optimal value and that $M>1$, when the diversity of the M samples of $\tilde{\mathcal{D}}$ is at its maximum, there exist no two individuals in $\tilde{\mathcal{D}}$ whose scaled distance Δ_{ij} is less than σ_{share}. Hence, it follows directly from (4.14) and (4.15) that $S_f(\Delta_{ij})=0$ when $i \neq j$. In this case, U attains its maximum value of 1, indicating maximum diversity. On the other hand, when the diversity of the M samples of $\tilde{\mathcal{D}}$ is at its minimum, all M individuals are identical and thus correspond to a single point in $\tilde{\mathcal{D}}$. Hence, $S_f(\Delta_{ij})=1$ for any solution pair since $\Delta_{ij}=0$ for all i and j. Therefore, $\sigma_i=M$ for all i, and the inner summation in Eq. (4.20) contributes exactly $M-1$ shares for every one of the M individuals. In this case, U attains its minimum value of zero, indicating minimum diversity. Note that the accuracy of diversity measure U depends on the optimality of σ_{share}.

Since no information about set $\tilde{\mathcal{D}}$ is used in Eq. (4.20), diversity measure U can only quantify the degree of uniformity of the M solutions within hypercuboid $\mathcal{H}$ whose size is defined by vector $\mathbf{h}$. This hypercuboid does not necessarily contain the entire set $\tilde{\mathcal{D}}$, and hence, its n-volume may be very small if $\tilde{\mathcal{D}}$ is not sampled adequately. Therefore, if the M solutions are near-uniformly distributed within $\mathcal{H}$, the value of U will approach unity (indicating good performance), although $\mathcal{H}$ may only occupy a very small region in $\tilde{\mathcal{D}}$. This problem can be overcome by considering the hypercuboid with the smallest n-volume that contains the *entire* set $\tilde{\mathcal{D}}$. The size of this hypercuboid is given by the following vector.

$$\mathbf{h}_{max}=\left[\max_{\mathbf{z}} z_1 \;\cdots\; \max_{\mathbf{z}} z_n\right]^T-\left[\min_{\mathbf{z}} z_1 \;\cdots\; \min_{\mathbf{z}} z_n\right]^T,\quad \mathbf{z}=\left[z_1 \cdots z_n\right]\in\tilde{\mathcal{D}} \tag{4.21}$$

Vectors $\mathbf{h}$ and $\mathbf{h}_{max}$ can be used to obtain an estimate of the amount of n-volume of $\tilde{\mathcal{D}}$ that is spanned by the M solutions. This estimate can be used to correct the value of U, resulting in the following, more accurate diversity measure.

$$U_0=U\frac{\prod_{i=1}^{n} h_i}{\prod_{i=1}^{n} h_{max,i}} \tag{4.22}$$

where $h_{max,i}$ is the i-th element of $\mathbf{h}_{max}$, and the numerator and denominator of the ratio in Eq. (4.22) are the n-volumes of the hypercuboids of size $\mathbf{h}$ and $\mathbf{h}_{max}$, respectively. It is easy to see that this ratio is always positive and cannot exceed unity. Therefore, U_0 is always bounded in the closed interval [0,1], with $U_0=1$ indicating maximum diversity over the entire set $\tilde{\mathcal{D}}$, and $U_0=0$ indicating minimum diversity.

Note that set $\tilde{\mathcal{D}}$ must be known *a priori*, in order to compute U_0. But knowing $\tilde{\mathcal{D}}$ means that the optimisation problem has already been solved. Therefore, U_0 can only be used for the evaluation of methods under test problems for which $\tilde{\mathcal{D}}$ is already known, such as the PI controller tuning problem shown in Fig. 4.6. In cases where diversity must be assessed without prior knowledge of $\tilde{\mathcal{D}}$, diversity measure U may be used.

4.5.2 Fitness Assignment Configurations

In order to evaluate the performance of adaptive fitness sharing and its combination with population ranking, six different fitness assignment configurations were considered in this work. Their flow charts are shown in Fig. 4.7.

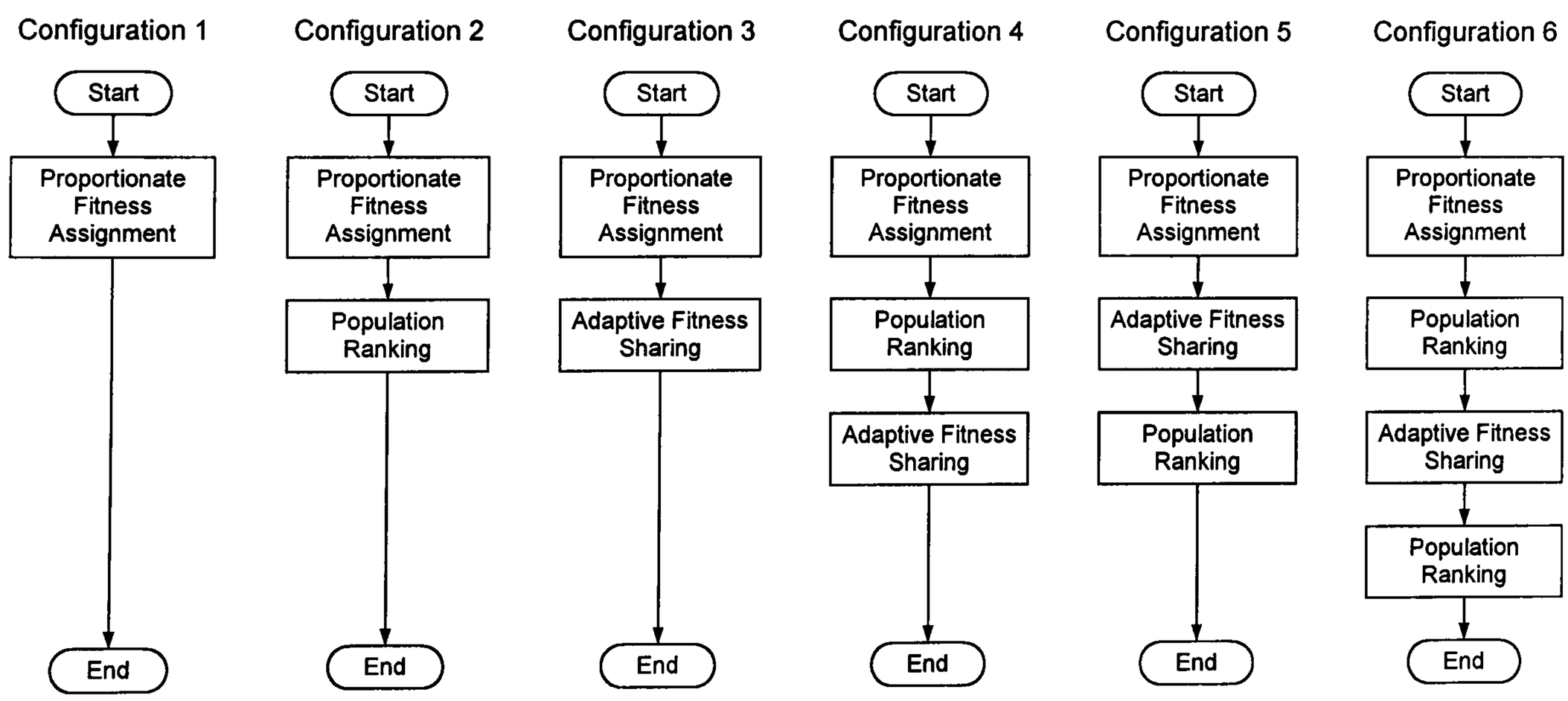


Fig. 4.7 *Flow charts of the six fitness assignment configurations*

Configuration 1 corresponds to a conventional, simple GA with proportionate fitness assignment, while configurations 2 to 6 also include population ranking, adaptive fitness sharing, and various combinations of the two.

4.5.3 Simulation Results – No Mutation

Fig. 4.8 shows the set of M solutions in $\tilde{\mathcal{D}}$, obtained by the GA at generation 100, using fitness assignment configuration 1 without mutation. It is observed that the distribution of solutions is far from uniform, and that there are large regions in $\tilde{\mathcal{D}}$ which are not sampled at all by the GA. This was expected, since no population ranking or adaptive fitness sharing is applied in this configuration.

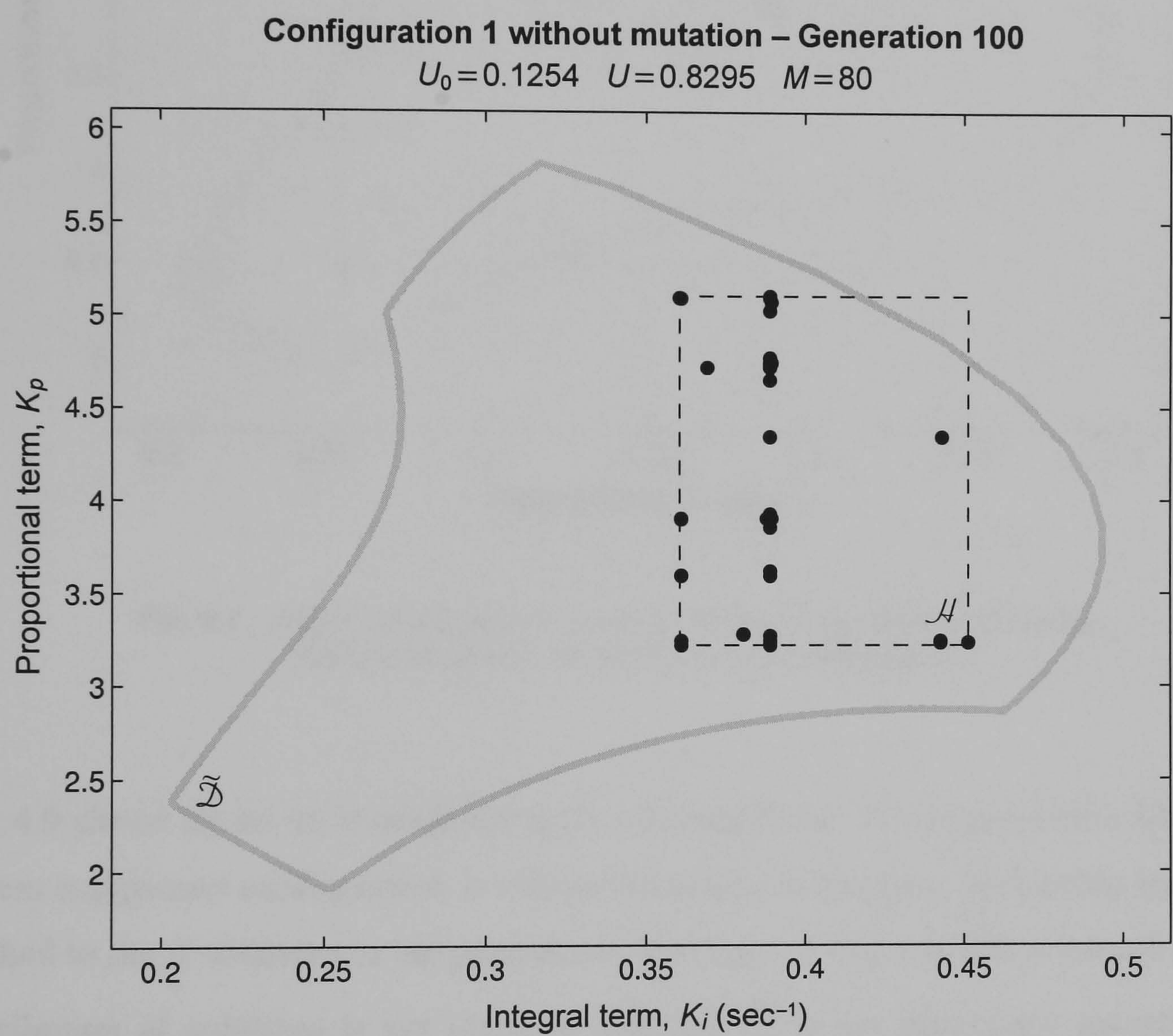


Fig. 4.8 *Map of strictly optimal solutions obtained at generation 100 using fitness assignment configuration 1 without mutation*

The inadequate sampling of $\tilde{\mathcal{D}}$ is also indicated by the size of parallelogram $\mathcal{H}$, whose area is much smaller than that of $\tilde{\mathcal{D}}$. Quantitatively, U_0=0.1254, which is very close to zero, indicating poor performance. In contrast to that, U=0.8295, which is much higher than the value of U_0. This is because U quantifies the degree of diversity only within $\mathcal{H}$, since no information about $\tilde{\mathcal{D}}$ is used in the calculation of U.

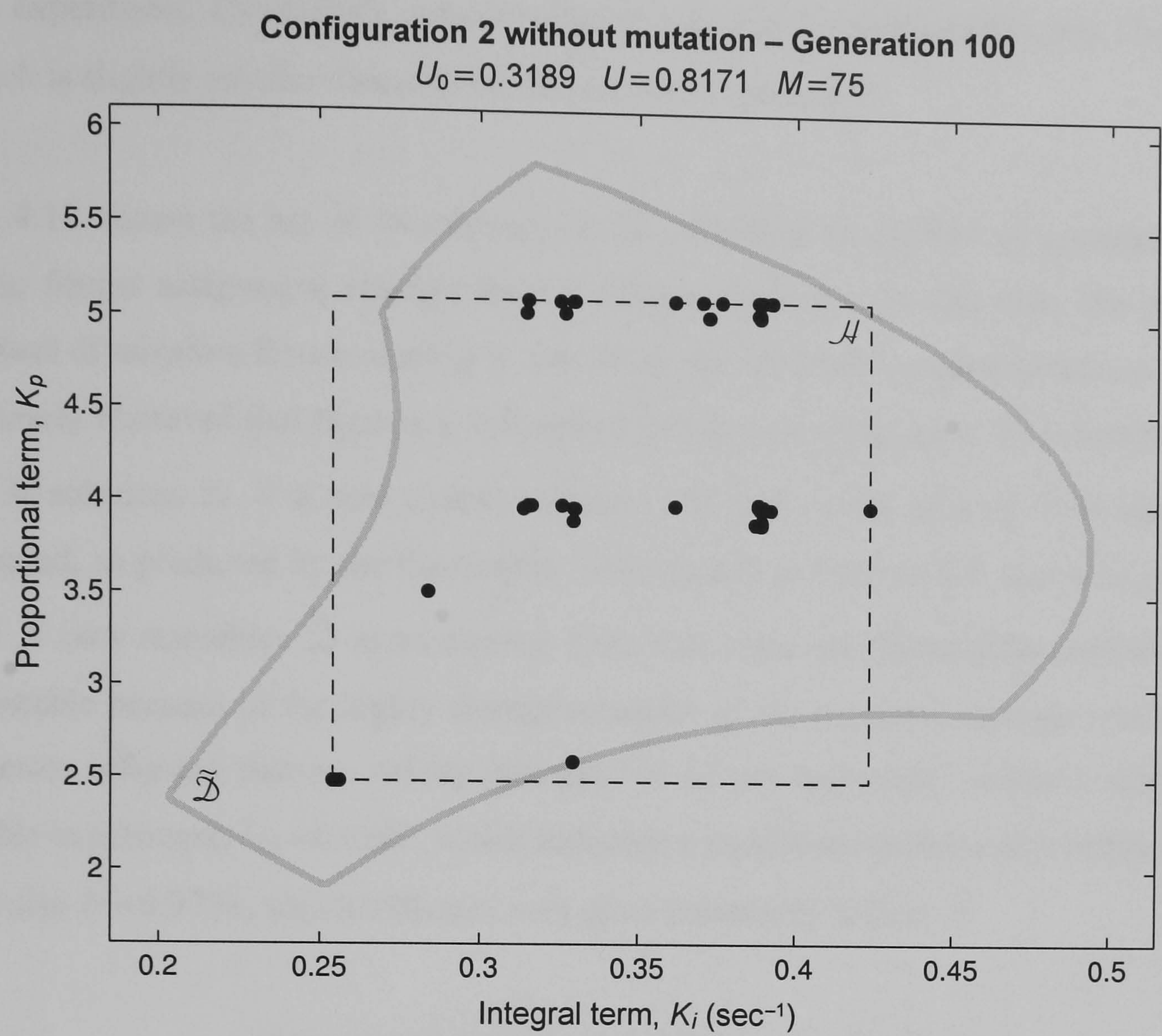


Fig. 4.9 *Map of strictly optimal solutions obtained at generation 100 using fitness assignment configuration 2 without mutation*

Fig. 4.9 shows the set of M solutions in $\tilde{\mathcal{D}}$, obtained by the GA at generation 100, using fitness assignment configuration 2 without mutation. In this case, population ranking is applied to the N solutions in the population. Similarly to Fig. 4.8, it is observed that the distribution of solutions is not uniform, and that $\tilde{\mathcal{D}}$ is not adequately sampled. The reason for this lack of improvement is because all M solutions in $\tilde{\mathcal{D}}$ have precisely the same objective function value of zero. Therefore, their reproductive potential *relative to each other* cannot be affected by population ranking. This clearly means that population ranking alone cannot improve the diversity of the M strictly optimal solutions. However, the diversity of the entire population of N solutions can be improved, especially in the initial stages of the GA run, where not many strictly optimal solutions have been found and $M \ll N$. For this reason, it is generally beneficial to employ population ranking in the fitness assignment strategy, but it is clear from Fig. 4.9 that other methods must also be employed, in order to achieve an acceptable degree of sample diversity within $\tilde{\mathcal{D}}$. In

this experiment, $U_0=0.3189$, which indicates an overall improvement, but $U=0.8171$, which is slightly smaller than that of the previous experiment.

Fig. 4.10 shows the set of M solutions in $\tilde{\mathcal{D}}$, obtained by the GA at generation 100, using fitness assignment configuration 3 without mutation. In this case, the proposed method of adaptive fitness sharing is applied to the M strictly optimal solutions in $\mathcal{H}$. It is clearly observed that there is a vast improvement in performance. The distribution of the M solutions in $\mathcal{H}$ is now almost uniform, and most of the area of $\tilde{\mathcal{D}}$ is adequately sampled, as predicted by the theoretical development in Section 4.4. It is also observed that $\mathcal{H}$ now resembles $\tilde{\mathcal{D}}$ more closely. Note that some deviations from uniformity are inevitable because of the highly disruptive nature of the crossover operator, which may generate offspring that are strictly optimal, but do not necessarily preserve uniformity. In this experiment, $U_0=0.6683$, which indicates a significant performance improvement, and also $U=0.9774$, which indicates very good uniformity within $\mathcal{H}$.

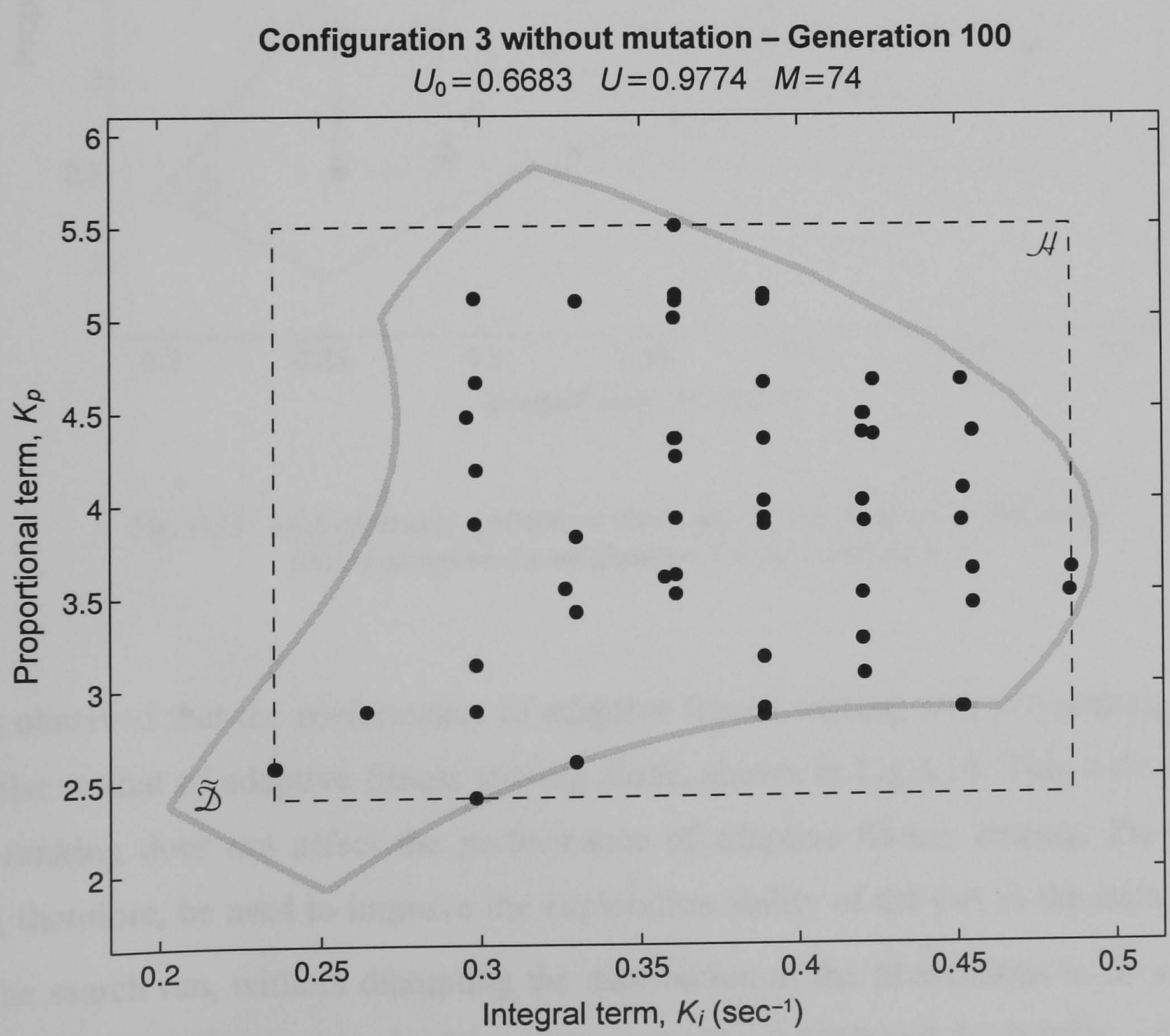


Fig. 4.10 *Map of strictly optimal solutions obtained at generation 100 using fitness assignment configuration 3 without mutation*

Fig. 4.11 shows the set of M solutions in $\tilde{\mathcal{D}}$, obtained by the GA at generation 100, using fitness assignment configuration 4 without mutation. In this case, population ranking is firstly applied to the N solutions in the entire population, and then adaptive fitness sharing is applied to the M strictly optimal solutions in $\mathcal{H}$. This configuration is called *adaptive fitness sharing with pre-ranking*.

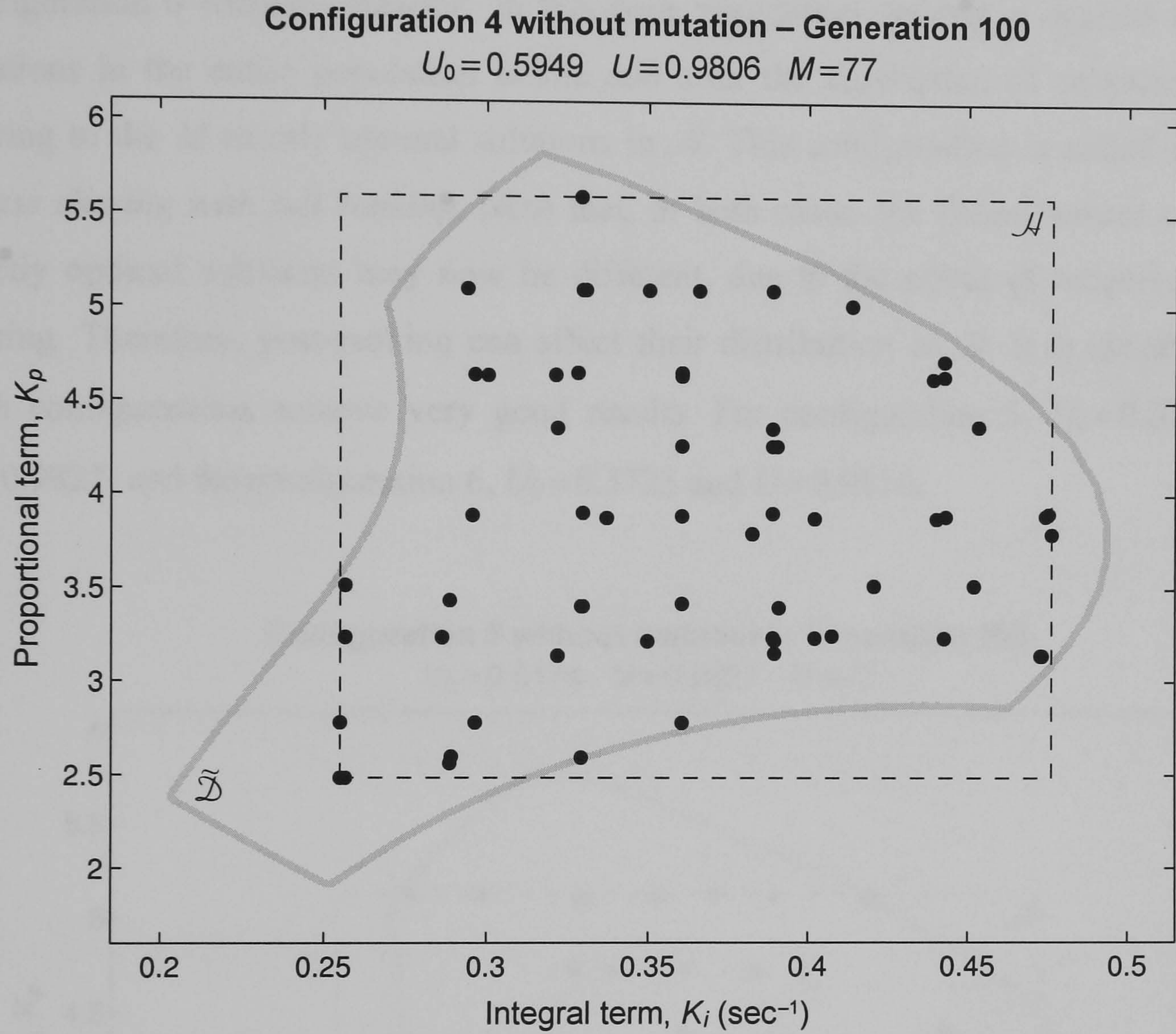


Fig. 4.11 *Map of strictly optimal solutions obtained at generation 100 using fitness assignment configuration 4 without mutation*

It is observed that the performance of adaptive fitness sharing with pre-ranking is very similar to that of adaptive fitness sharing alone, shown in Fig 4.10. This indicates that pre-ranking does not affect the performance of adaptive fitness sharing. Pre-ranking can, therefore, be used to improve the exploration ability of the GA in the initial stages of the search run, without disrupting the distribution of the M solutions in $\tilde{\mathcal{D}}$ achieved by adaptive fitness sharing. In this experiment, U_0=0.5949 and U=0.9806, which are similar to those obtained using adaptive fitness sharing alone.

Fig. 4.12 shows the set of M solutions in $\tilde{\mathcal{D}}$, obtained by the GA at generation 100, using fitness assignment configuration 5 without mutation. In this case, adaptive fitness sharing is firstly applied to the M strictly optimal solutions in $\mathcal{H}$, and then population ranking is applied to the N solutions in the entire population. This configuration is called *adaptive fitness sharing with post-ranking*. Finally, Fig. 4.13 shows the set of M solutions in $\tilde{\mathcal{D}}$, obtained by the GA at generation 100, using fitness assignment configuration 6 without mutation. In this case, population ranking is applied to the N solutions in the entire population before and after the application of adaptive fitness sharing to the M strictly optimal solutions in $\mathcal{H}$. This configuration is called *adaptive fitness sharing with full ranking*. Note that, in both cases, the fitness values of the M strictly optimal solutions may now be different, due to the effect of adaptive fitness sharing. Therefore, post-ranking can affect their distribution in $\mathcal{H}$. It is observed that both configurations achieve very good results. For configuration 5, $U_0=0.5174$ and $U=0.9827$, and for configuration 6, $U_0=0.5723$ and $U=0.9816$.

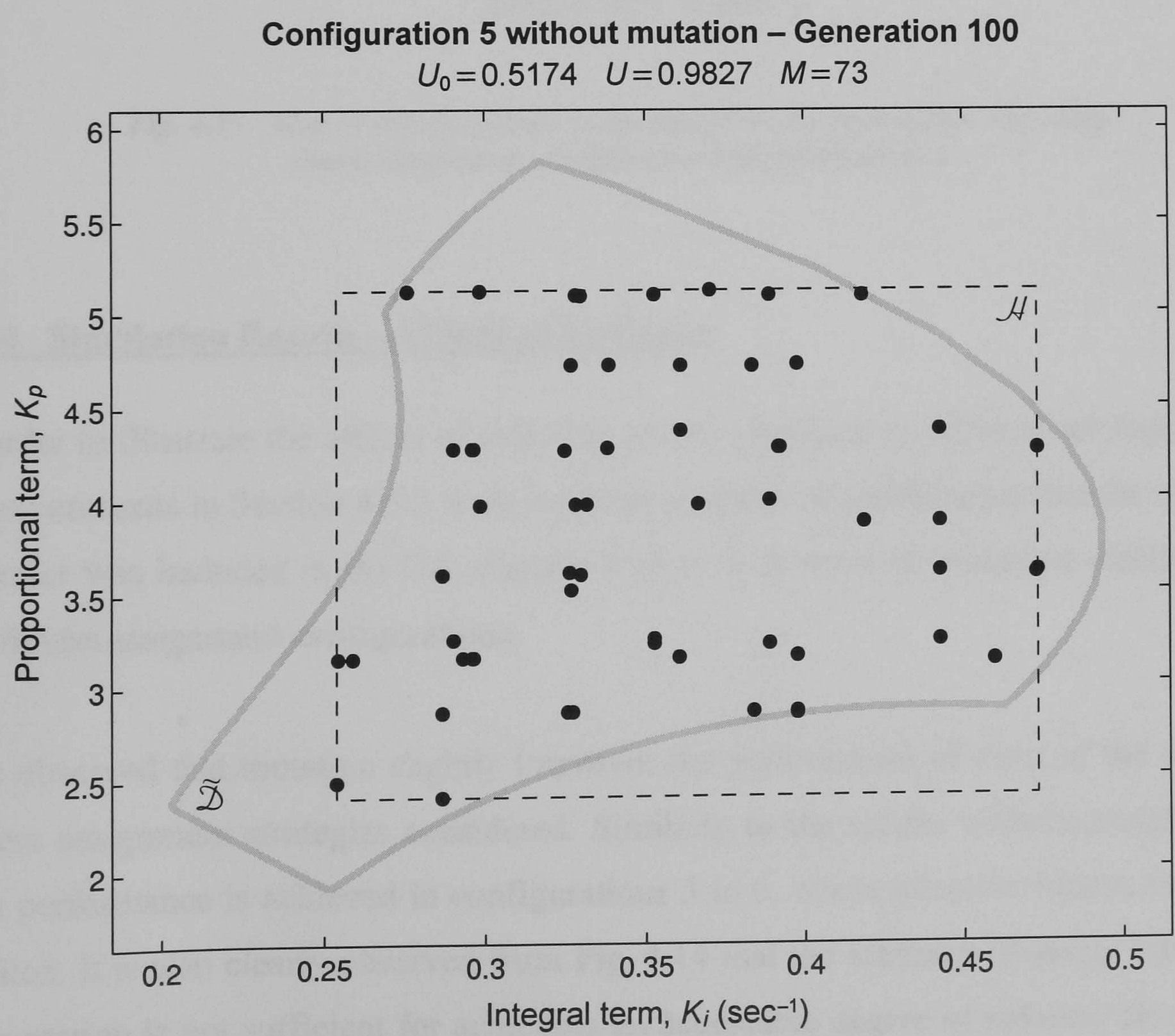


Fig. 4.12 *Map of strictly optimal solutions obtained at generation 100 using fitness assignment configuration 5 without mutation*

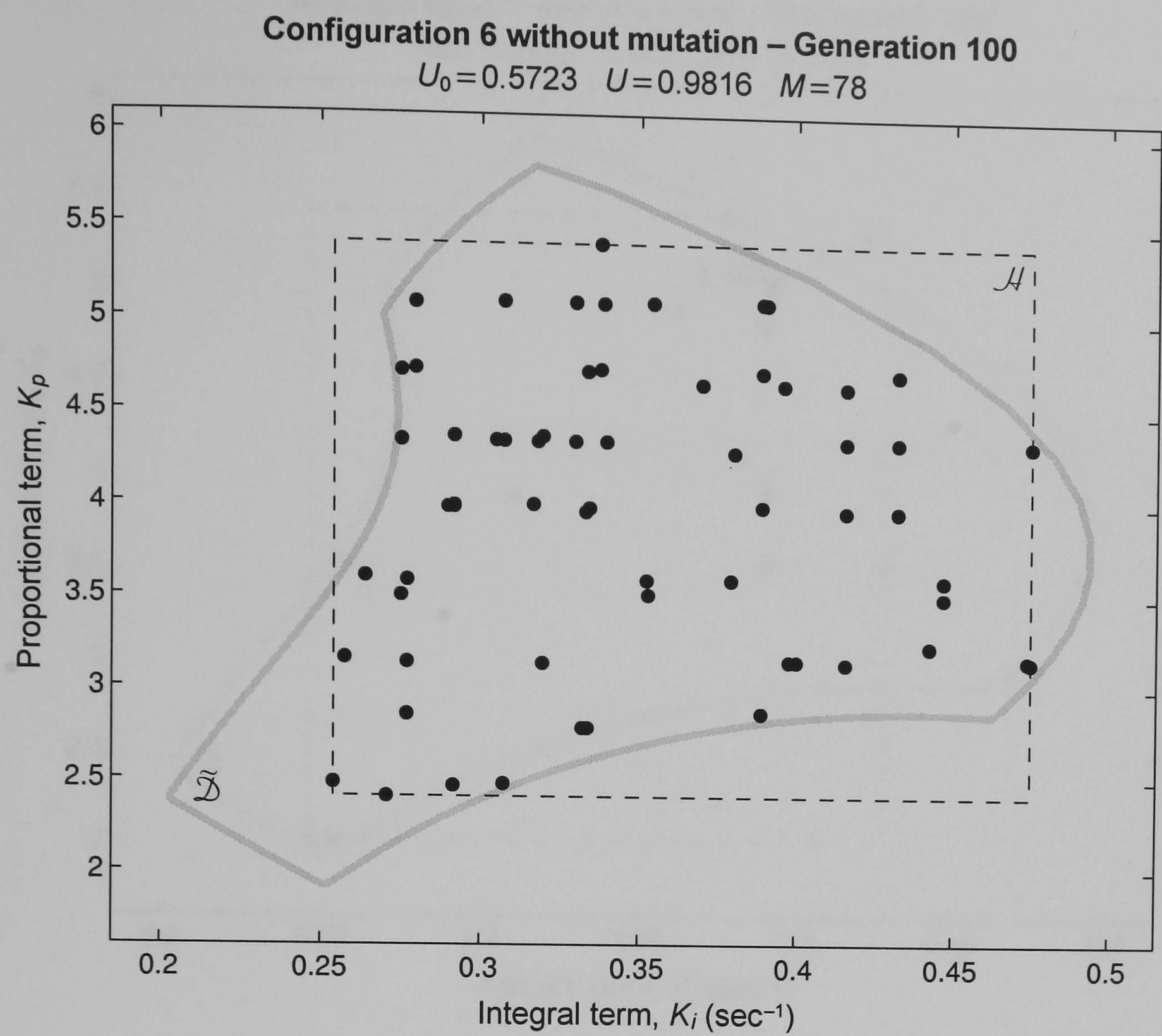
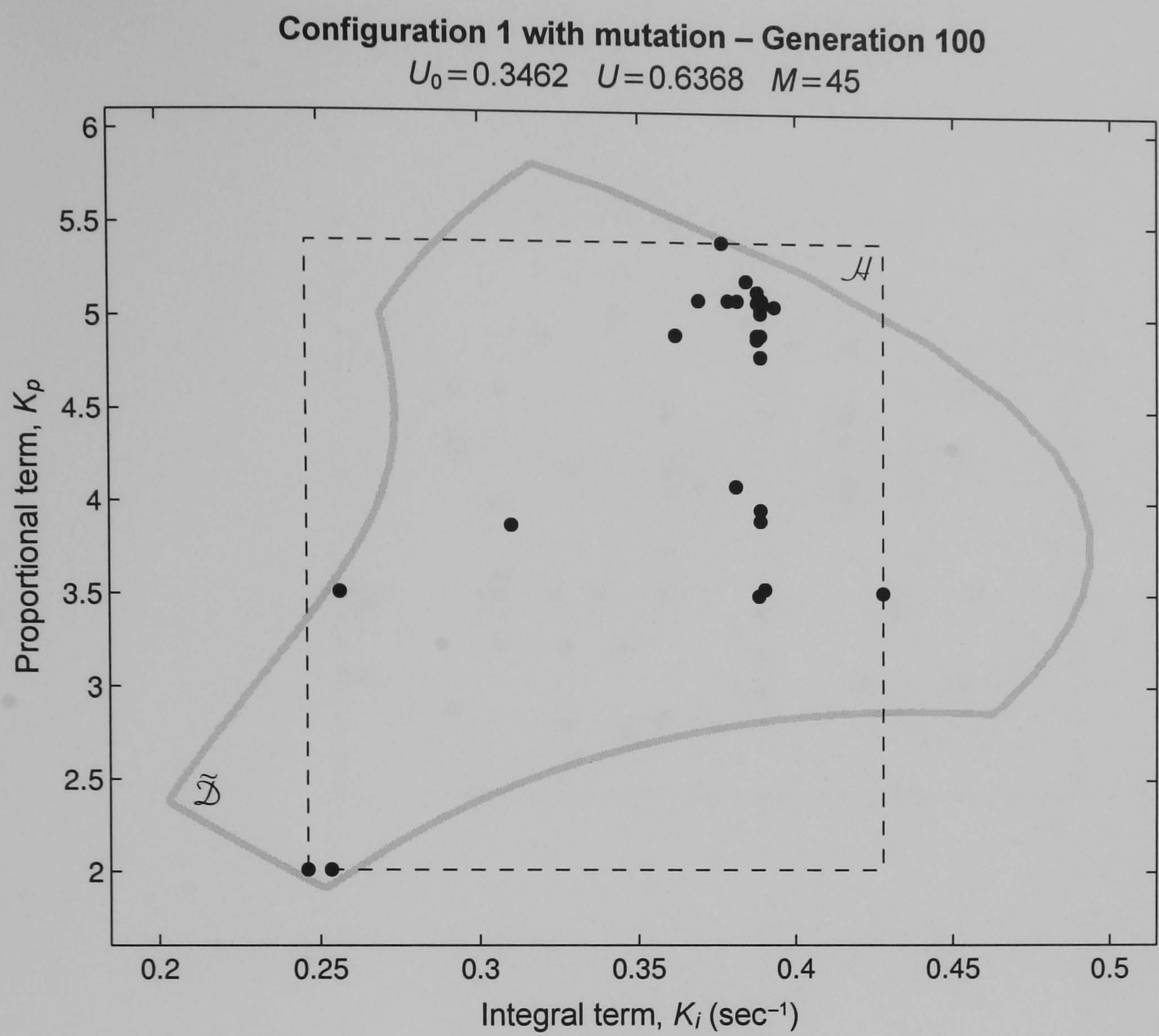


Fig. 4.13 *Map of strictly optimal solutions obtained at generation 100 using fitness assignment configuration 6 without mutation*

4.5.4 Simulation Results – Effects of Mutation

In order to illustrate the effects of mutation on the distribution of the M solutions in $\tilde{\mathcal{D}}$, all experiments in Section 4.5.3 were repeated with the only difference that the mutation operator was included in the GA. Figures 4.14 to 4.16 show the obtained results for all six fitness assignment configurations.

It is observed that mutation slightly improves the performance of most of the different fitness assignment strategies considered. Similarly to the results without mutation, the best performance is achieved in configurations 3 to 6, where adaptive fitness sharing is applied. It is also clearly observed from Fig. 4.14 that the arbitrary diversity introduced by mutation is not sufficient for achieving an acceptable degree of solution diversity in cases where adaptive fitness sharing is not applied, such as configurations 1 and 2. This clearly demonstrates the effectiveness of the proposed diversification method.



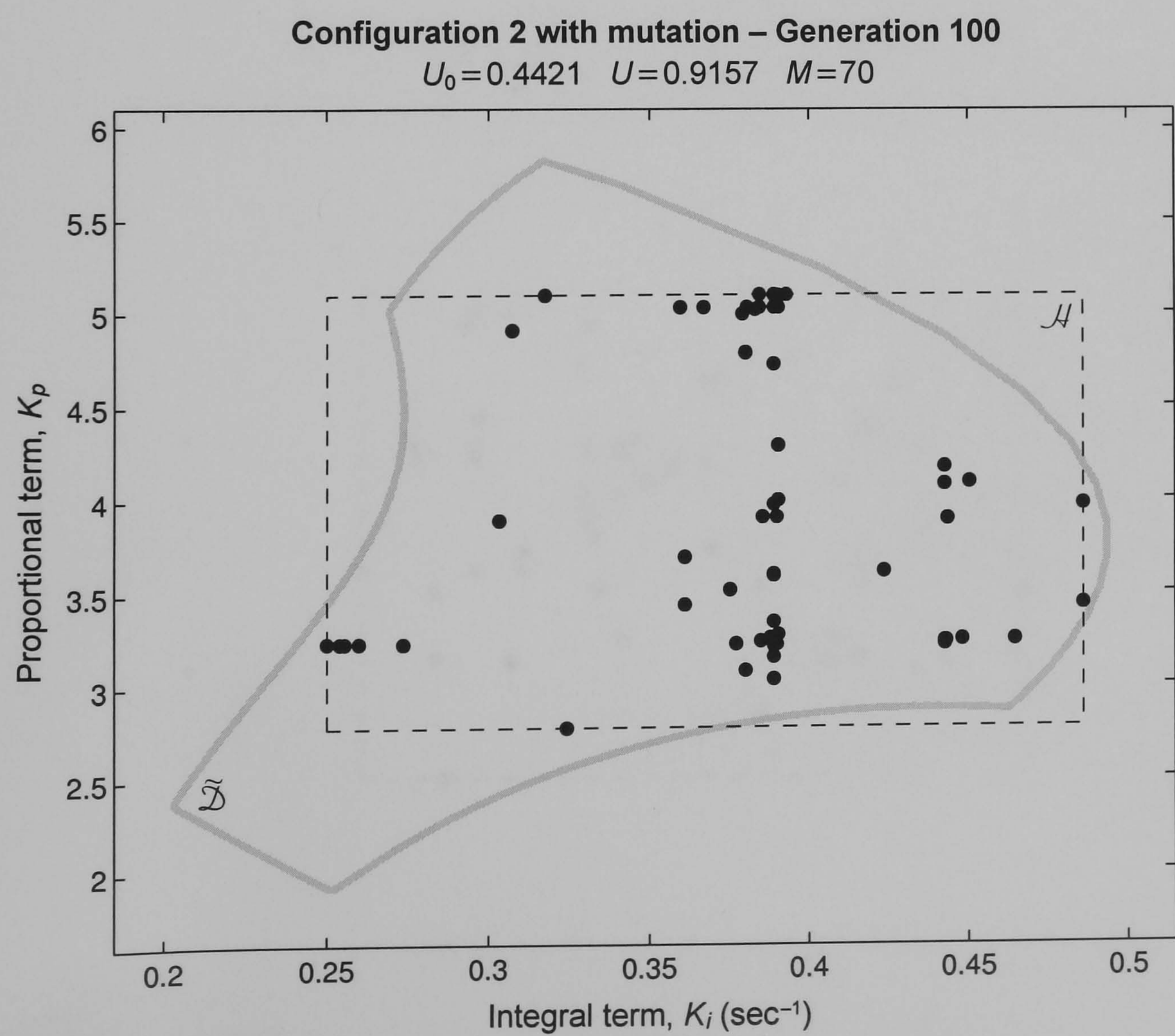
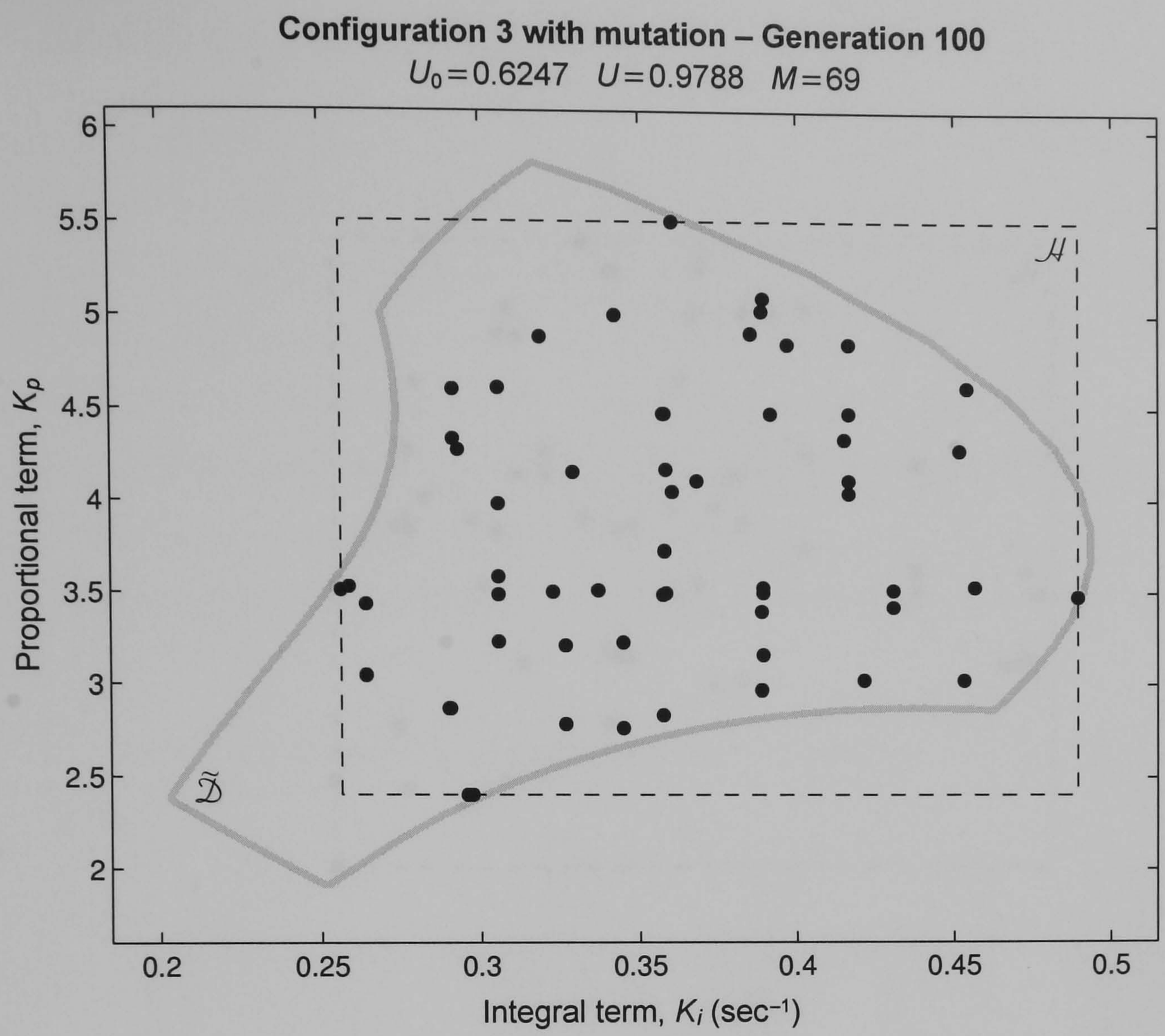


Fig. 4.14 *Effects of mutation on the performance of configurations 1 and 2*



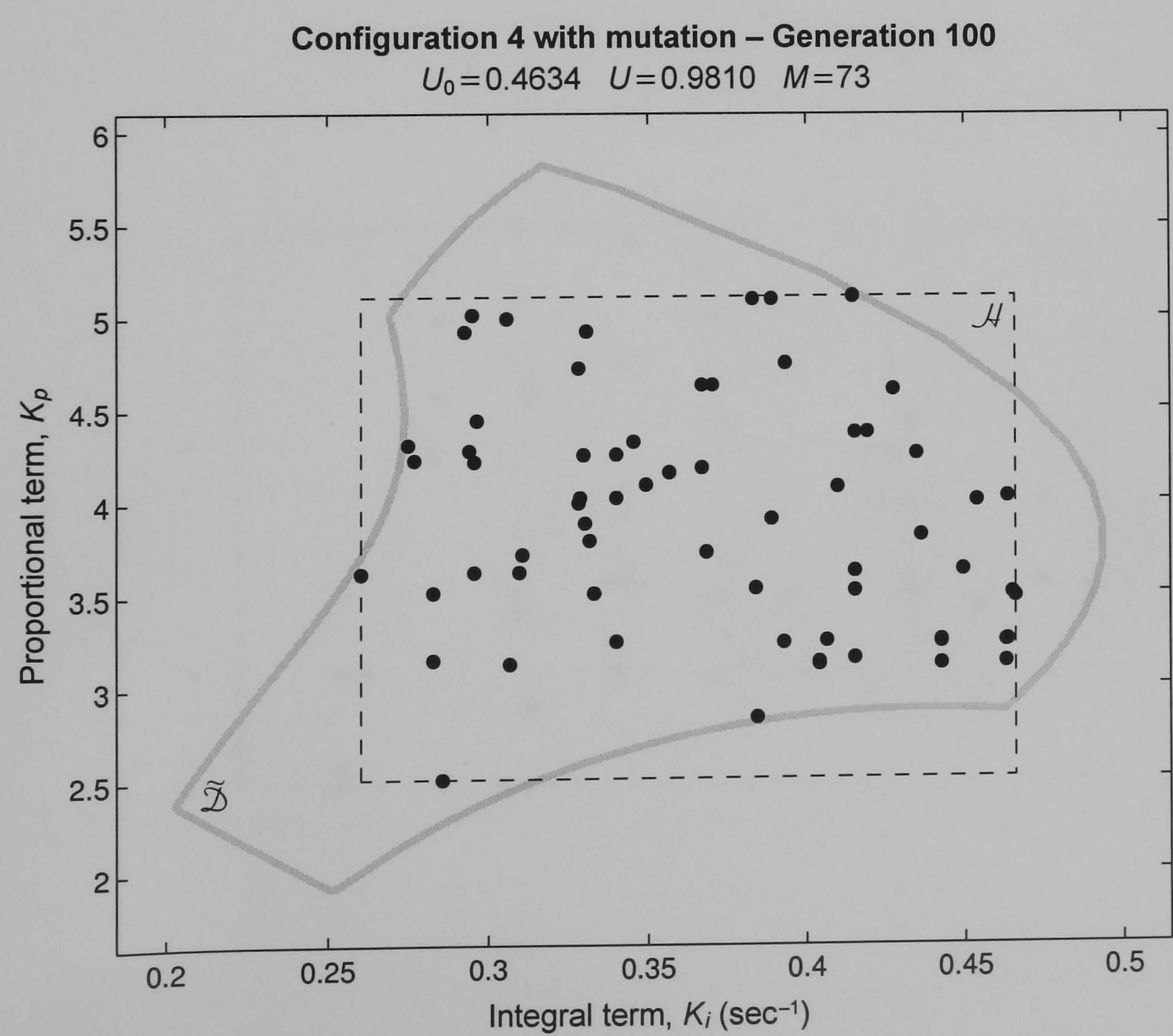
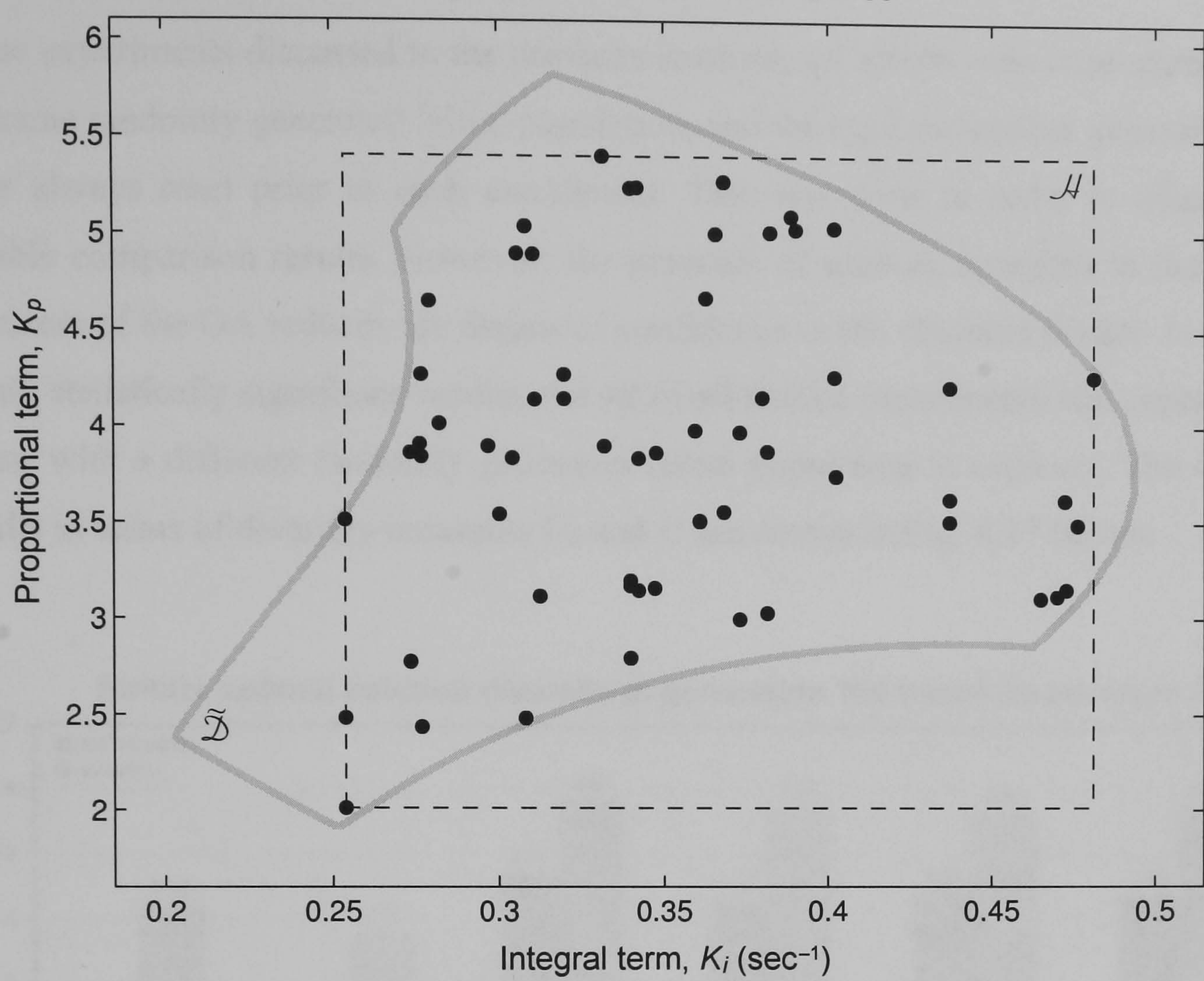


Fig. 4.15 Effects of mutation on the performance of configurations 3 and 4



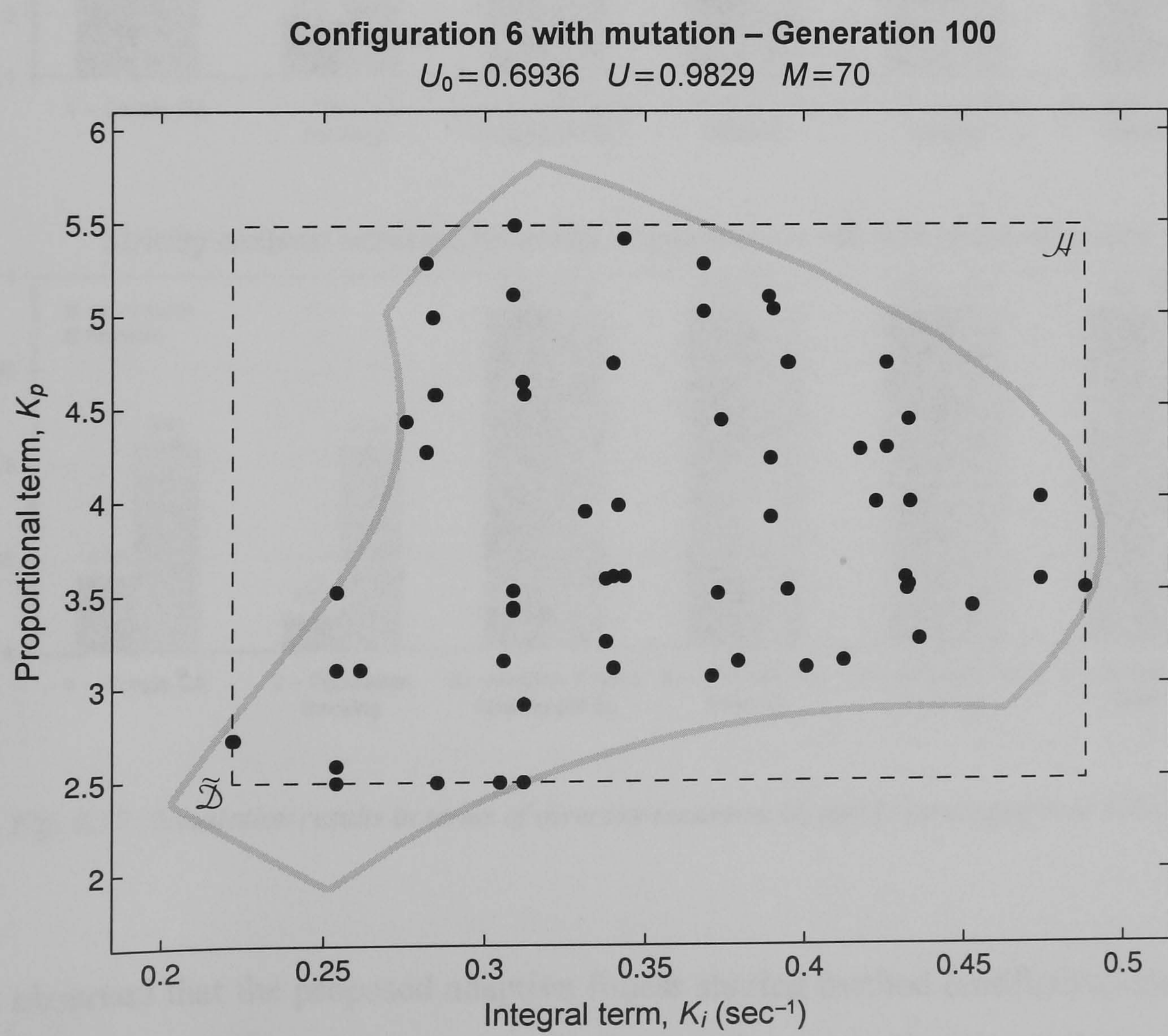


Fig. 4.16 *Effects of mutation on the performance of configurations 5 and 6*

4.5.5 Simulation Results – Statistical Tests

In the experiments discussed in the previous sections, all search runs were started using the same randomly generated initial population, and the random number generator seeds were always reset prior to each experiment. This was done in order to obtain more reliable comparison results. However, the presence of random elements in the generic operators of the GA reduces the degree of confidence in the obtained results. In order to obtain statistically significant results, the set of all twelve experiments was repeated 100 times, with a different randomly generated initial population in each set. The obtained results in terms of diversity measures U_0 and U are shown in Fig. 4.17 below.

Fig. 4.17 *Simulation results in terms of diversity measures U_0 and U (averaged over 100 runs)*

It is observed that the proposed adaptive fitness sharing method (configurations 3 to 6) consistently outperforms both the simple GA (configuration 1) and population ranking alone (configuration 2), in terms of both U_0 and U. In terms of diversity measure U_0, the

observed performance improvement using mutation is because of the dependence of U_0 on the size of $\mathcal{H}$. This can result in higher values of U_0 whenever mutation generates new solutions in $\tilde{\mathcal{D}}$ that result in an increase in the size of $\mathcal{H}$, even if these solutions actually disrupt the uniformity in $\mathcal{H}$. On the other hand, the value of diversity measure U does not depend on the size of $\mathcal{H}$. Hence, U quantifies the degree of uniformity of the solutions within $\mathcal{H}$, irrespective of its size relative to that of $\tilde{\mathcal{D}}$. In terms of diversity measure U, it is observed that adaptive fitness sharing achieves near-perfect results that are virtually unaffected by mutation. As expected, the degree of diversity achieved by the simple GA and population ranking alone is improved by mutation, but is still not sufficiently high, as illustrated in Figures 4.14 to 4.16.

The simulation results in terms of the number of strictly optimal solutions, M, contained in the population at generation 100, are shown in Fig. 4.18 below. It is observed that the simple GA is greatly affected by mutation, whereas all other methods maintain most of the population inside $\tilde{\mathcal{D}}$ even in the presence of mutation. It is also observed that the combinations of population ranking with adaptive fitness sharing achieve slightly higher values of M than adaptive fitness sharing alone.

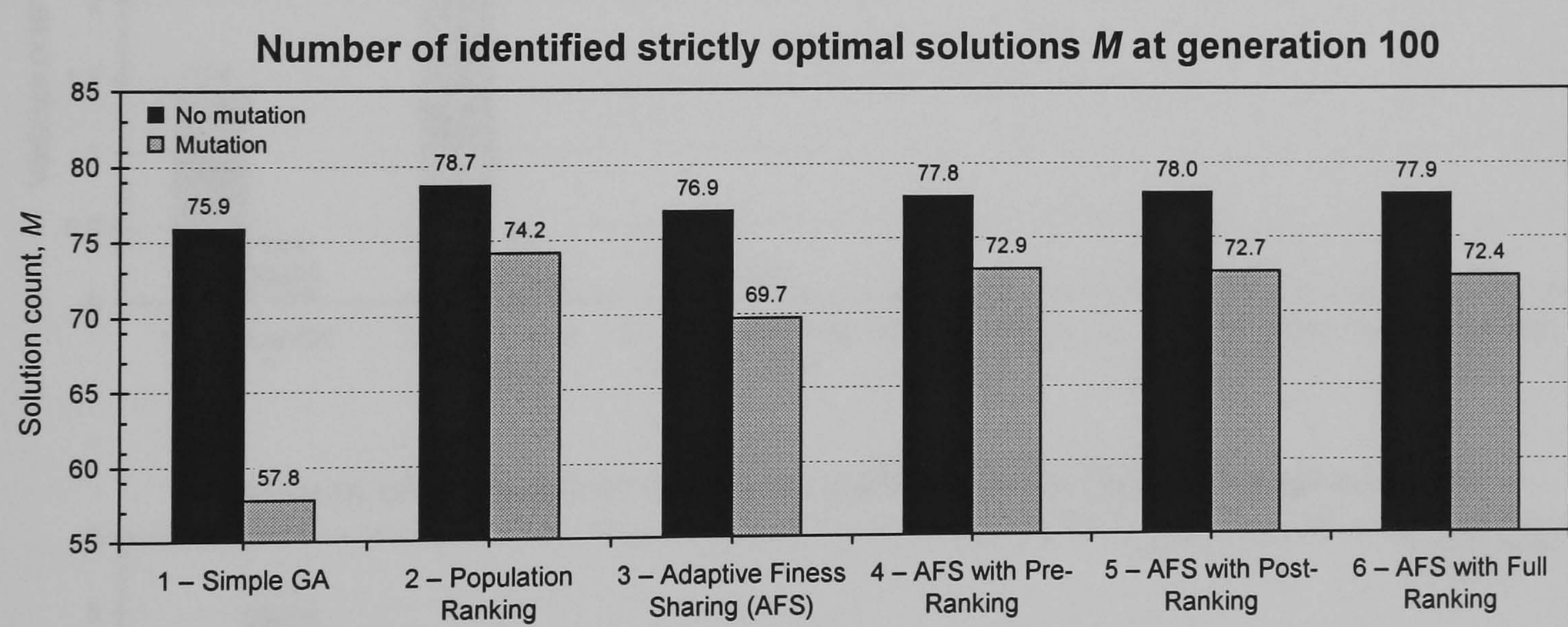


Fig. 4.18 *Number of identified strictly optimal solutions M (averaged over 100 runs)*

Assuming that no knowledge about the search landscape is known *a priori*, the initial population used in the beginning of a GA run is usually chosen at random. Therefore, in general, the performance of a given method is expected to change under different initial populations. Methods that are reasonably insensitive to changes in the initial conditions

are said to be *robust*. Robustness is a characteristic of fundamental importance in the assessment of methods containing random processes, such as GA-based methods. High robustness ensures a high degree of repeatability and increases the degree of confidence in the obtained results. In the case of the six fitness assignment configurations tested in this work, robustness can be assessed by examining the *variances* of the different results obtained from the 100 sets of experiments. These are shown in Fig. 4.19 below.

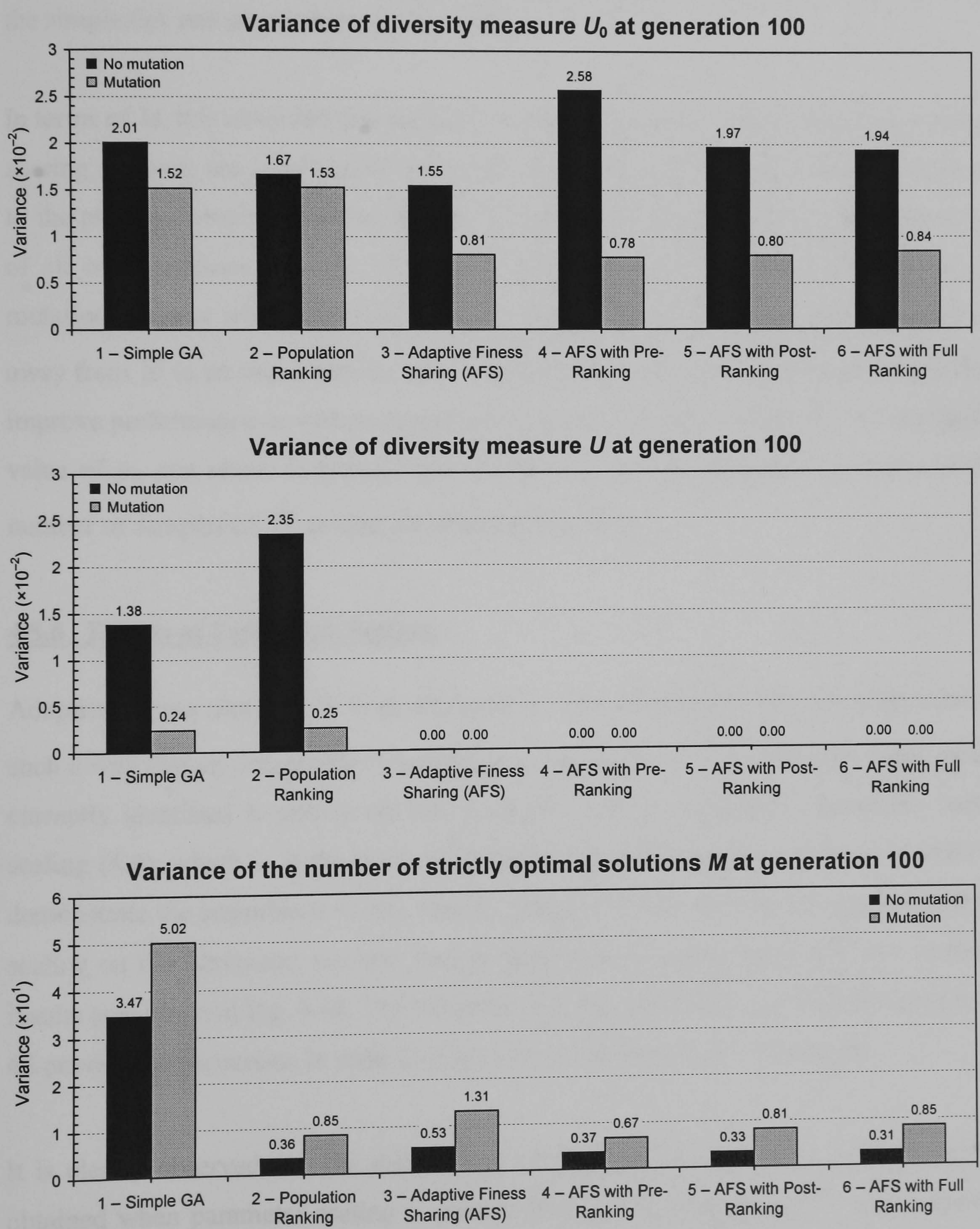


Fig. 4.19 *Variances of U_0, U, and M, obtained from the 100 sets of experiments*

In terms of U_0, it is observed that when no mutation is applied, adaptive fitness sharing alone (configuration 3) has the highest robustness, since it has the lowest variance of all six configurations. Mutation is shown to improve the robustness of all configurations, with the ones associated with adaptive fitness sharing having the highest robustness. In terms of U, it is observed that all four adaptive fitness sharing variants have excellent robustness that is almost unaffected by the absence of mutation, achieving virtually zero variance in all cases. Mutation is again shown to improve robustness, especially that of the simple GA and population ranking alone.

In terms of M, it is observed that population ranking alone, as well as all adaptive fitness sharing variants, are significantly more robust than the simple GA. Note that, contrary to the previous observations for U_0 and U, mutation is shown to reduce the robustness of all configurations in terms of M. This is because of the disruptive nature of the mutation operator which can easily destroy strictly optimal solutions, by moving them away from $\tilde{\mathcal{D}}$ in an unpredictable way. It is, therefore, concluded that mutation can help improve performance as well as robustness in terms of solution diversity, but too high a value of p_m can result in reduced performance as well as robustness in terms of the number of samples of $\tilde{\mathcal{D}}$, as also illustrated in Fig. 4.18.

4.5.6 Effects of Parameter Scaling

Adaptive fitness sharing has been designed in order to distribute the GA population in such a way that an *acceptable visualisation* of the shape of $\tilde{\mathcal{D}}$ is obtained, based on the currently identified M strictly optimal solutions. This is achieved by parameter vector scaling (4.9), which is at the heart of the proposed fitness sharing method. In order to demonstrate the importance of this scaling, adaptive fitness sharing was applied with no scaling on the parameter vectors. This is equivalent to setting $\mathbf{h}=[1\ 1]^T$. The obtained results are shown in Fig. 4.20. The initial state of the population was kept the same as in all previous experiments, in order to obtain more reliable comparison results.

It is clearly observed that the distribution of the M solutions in $\tilde{\mathcal{D}}$ is inferior to that obtained when parameter scaling is present (Fig. 4.12). This is because of the uneven effect of fitness sharing in the horizontal and vertical directions, caused by the lack of

scaling. This can be better understood by considering the parallelogram marked ▨ in Fig. 4.20. Solutions in $\tilde{\mathcal{D}}$ that are all contained in the interior of a single parallelogram of this size will share their fitness values. It is observed that the aspect ratio of this parallelogram is significantly different from that of $\mathcal{H}$. In the horizontal direction, sharing penalises individuals that are considerably distant with respect to the size of $\tilde{\mathcal{D}}$. Conversely, in the vertical direction, individuals that are very close to each other are not penalised at all. It can be seen that this causes the M solutions to cluster around specific regions, forming vertical 'columns' of distance σ_{share}. The location of these regions is unpredictable and is affected by the initial state of the population. It should be stressed here that a mere scaling of the parameter vectors with respect to the size of the search space will not, in general, achieve acceptable results. It is the size of $\mathcal{H}$, given in **h**, that must be used to scale the parameters, because the aspect ratio of $\mathcal{H}$ can be considerably different from that of the entire search space. Scaling the parameters using **h** also gives the proposed method its adaptive properties, as demonstrated in the following section.

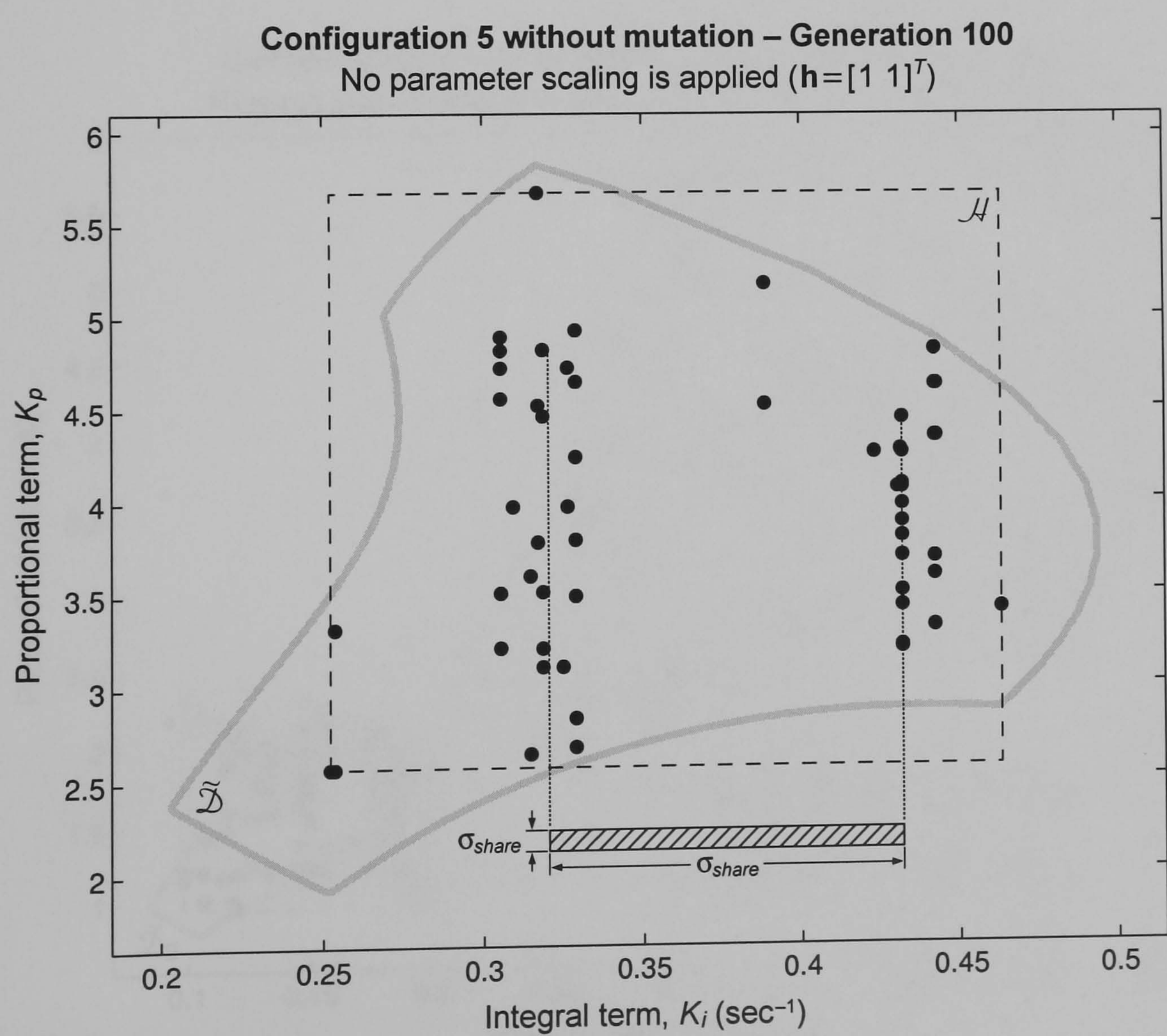


Fig. 4.20 *Effects of parameter vector scaling on the distribution of solutions in* $\tilde{\mathcal{D}}$

4.5.7 Adaptation of the Population Density

Adaptive fitness sharing has been designed in order to achieve a population distribution density that is kept constant relative to the size of $\tilde{\mathcal{D}}$. This means that in cases where the size of $\tilde{\mathcal{D}}$ changes during the course of a GA run, the density of the population also changes accordingly. The sharing radius σ_{share} itself does not change, but its optimality is maintained by dynamically changing the scaling of the parameter vectors. In order to demonstrate this, an experiment was performed in which the gain K of process $G(s)$ in Fig. 4.6 was changed during the course of the GA run. At generation 50, K was changed from its original value of $K_1=0.5$ to its new value of $K_2=1.2$. This modifies the search landscape and causes $\tilde{\mathcal{D}}$ to change. Let $\tilde{\mathcal{D}}_1$ denote set $\tilde{\mathcal{D}}$ when $K=K_1$, and $\tilde{\mathcal{D}}_2$ denote set $\tilde{\mathcal{D}}$ when $K=K_2$. The gain change causes $\tilde{\mathcal{D}}$ to retain its shape, but shrink in size by the ratio K_1/K_2. Its location also changes, so that $\tilde{\mathcal{D}}_1$ and $\tilde{\mathcal{D}}_2$ are disjoint sets. The obtained results at generations 50 and 100 are shown in Fig. 4.21 below.

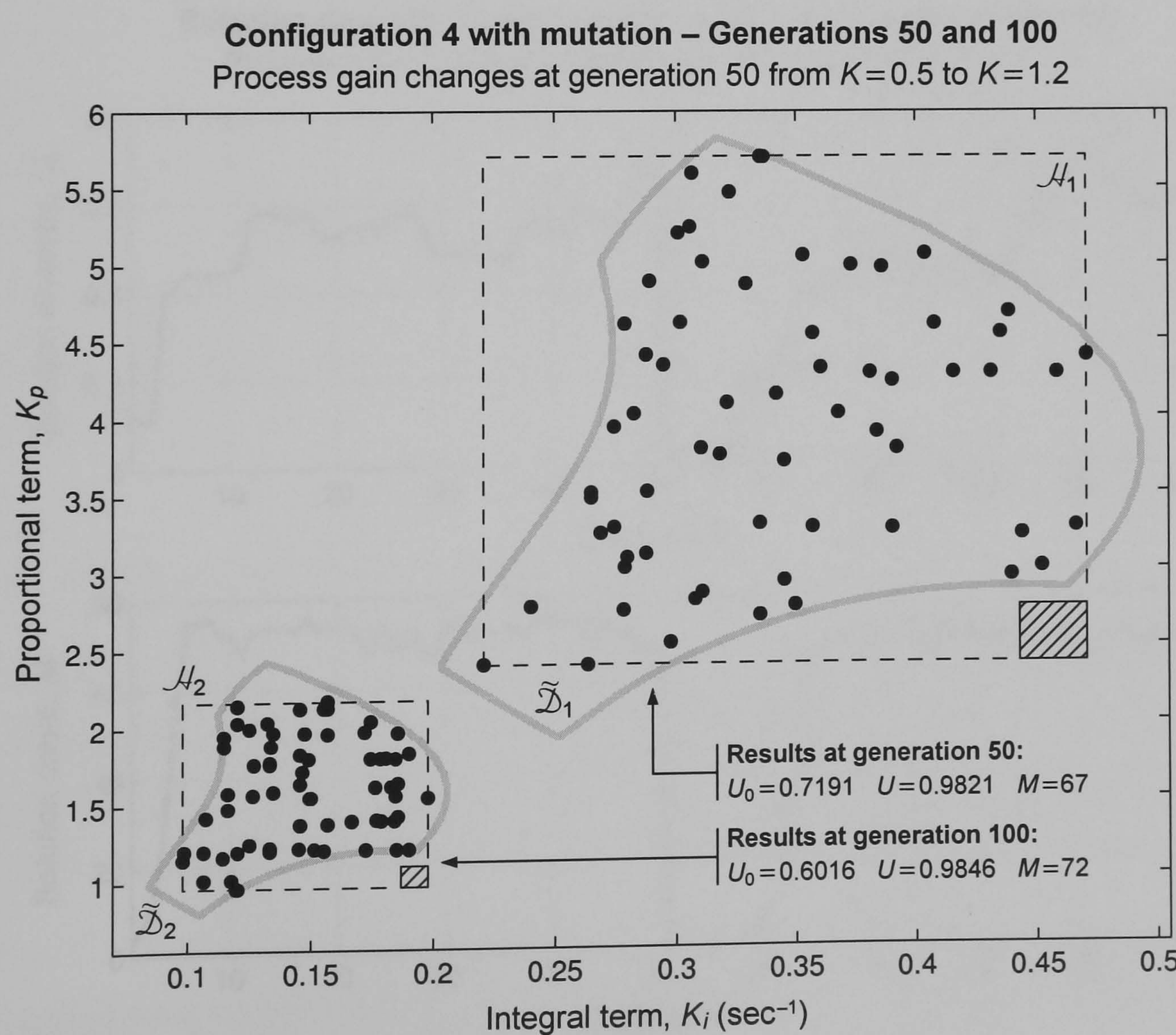


Fig. 4.21 *Adaptation of the population density using adaptive fitness sharing*

It is observed that the distribution of solutions in $\tilde{\mathcal{D}}_1$ at generation 50 is near-uniform, as indicated by diversity measures U_0 and U. The parallelogram marked ▨ that is located inside $\mathcal{H}_1$ indicates the size of the spatial neighbourhood where fitness sharing occurs. Because of scaling (4.9), the size of this parallelogram is given by vector $\mathbf{h}\sigma_{share}$, and its aspect ratio is thus the same as that of $\mathcal{H}_1$. At generation 50, most of the N individuals have successfully converged inside $\tilde{\mathcal{D}}_1$ as indicated by the value of M. After the process gain change at generation 50, these M individuals are no longer strictly optimal, and the population is expected to move away from $\tilde{\mathcal{D}}_1$ and converge in $\tilde{\mathcal{D}}_2$. This is precisely what was observed. As can be seen in Fig. 4.21, at generation 100 the majority of the population has successfully converged inside $\tilde{\mathcal{D}}_2$, and has a near-uniform distribution as indicated by the new values of U_0 and U. Notice that, although the value of σ_{share} was never changed during the entire GA run, the population at generation 100 has adapted to the required higher density, since the size of $\tilde{\mathcal{D}}_2$ is smaller than that of $\tilde{\mathcal{D}}_1$.

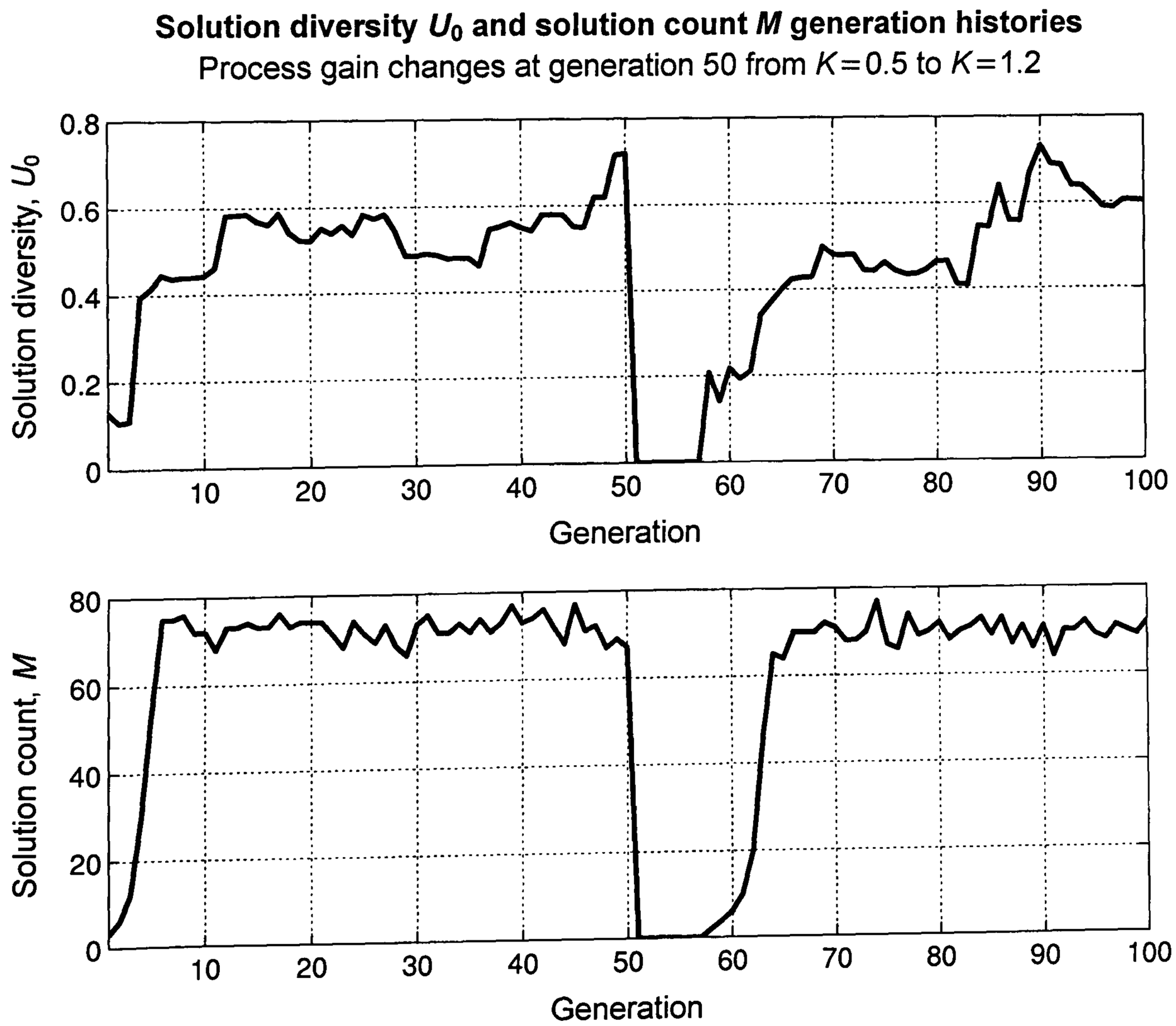


Fig. 4.22 *Generation histories of solution diversity measure U_0 and solution count M*

Fig. 4.22 shows the generation time histories of U_0 and M throughout the entire GA run. In the beginning of the search run, it is observed that the sample diversity as well as the number of samples of $\tilde{\mathcal{D}}_1$ rapidly increase in a few generations. When the process gain change occurs at generation 50, solution count M drops to zero and stays there for a few generations. This is because the M solutions in $\tilde{\mathcal{D}}_1$ do not belong to $\tilde{\mathcal{D}}_2$ and hence they are no longer strictly optimal. After a number of generations, both U_0 and M increase and reach similar levels to those prior to the gain change.

An interesting observation is that both U_0 and M increase much faster in the beginning of the GA run, than after the gain change occurs. This is because the population in the beginning of the GA run was chosen at random, and hence, the GA was very rich in genetic material to guide the search towards $\tilde{\mathcal{D}}_1$. On the other hand, immediately after the gain change, 67 of the total 80 individuals in the population were all located in the region occupied by set $\tilde{\mathcal{D}}_1$, which is only a small subset of the entire search space. This limits the amount of genetic material in the population, and inevitably slows down convergence. The presence of mutation in such cases can play an important role, by introducing new genetic material that cannot be generated by crossover alone. Based on the above justifications, it may be beneficial to slightly increase the mutation probability p_m in cases where large changes in the search landscape are expected. Furthermore, it may be beneficial to monitor M at each generation and either modify p_m as required, or randomly re-initialise a proportion of the population whenever the value of M becomes very low or zero.

4.5.8 Simulation Results in the Time Domain

The time-domain responses of the strictly optimal solutions obtained using a simple GA and adaptive fitness sharing with full ranking are shown in Fig. 4.23. These results were obtained with the mutation operator present, and thus correspond to the maps shown in Fig. 4.14 (top) and Fig. 4.16 (bottom), respectively. As expected, it is observed that the degree of uniformity in $\tilde{\mathcal{D}}$ is reflected in the spread of the time-domain responses inside the performance specification envelope. Adaptive fitness sharing clearly achieves better results, with responses having peak overshoots ranging from 0% up to the maximum requirement of 20%, and settling times of at most 25 sec, as required.

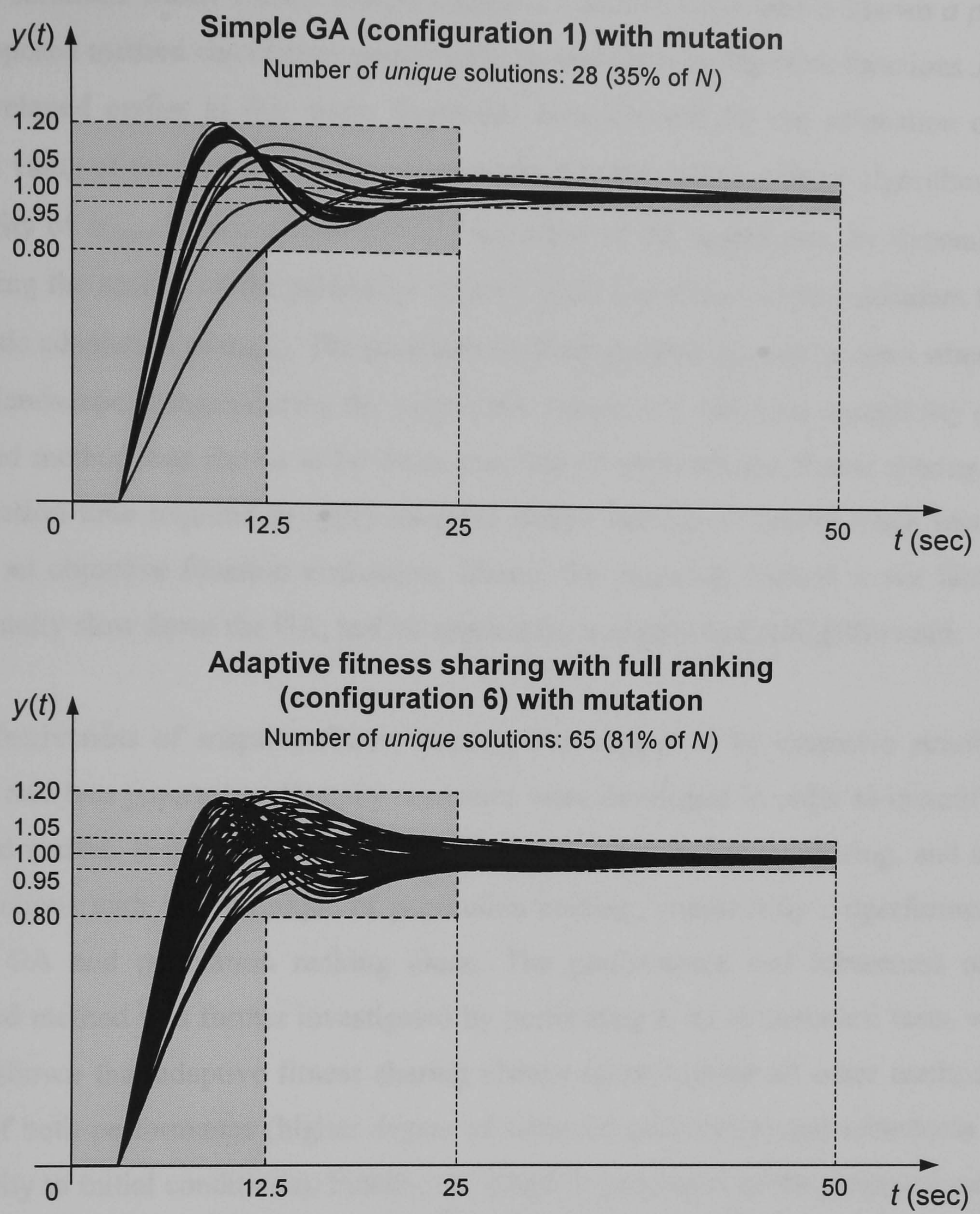


Fig. 4.23 *Closed-loop system responses of the M solutions in $\tilde{\mathcal{D}}$ obtained using a simple GA (top) and adaptive fitness sharing (bottom)*

4.6 Summary

In this chapter, a new method called *adaptive fitness sharing* was proposed, whose purpose is to enable a GA to locate multiple equivalent optimal solutions and distribute the members of the population uniformly within the optimal solution set. The proposed method is based on the techniques of niche formation and speciation, and is applicable to the optimisation of search landscapes which contain an infinite number of equivalent

optimal solutions which share a unique objective function value that is known *a priori*. The proposed method can thus be used for the optimisation of objective functions J_S and J_M, developed earlier in this work. Formulae were derived for the estimation of the optimal value of the sharing radius σ_{share} involved in the fitness sharing algorithm. The optimality of σ_{share} is maintained during the course of the search run, by dynamically modifying the scaling of the parameter vectors. This was shown to be equivalent to the automatic adaptation of σ_{share}. The proposed method can thus be used in cases where the search landscape changes during the course of a search run. The time complexity of the proposed method was shown to be better that that of conventional fitness sharing. The computation time required to apply adaptive fitness sharing is usually much less than that of an objective function evaluation. Hence, the proposed method is not likely to significantly slow down the GA, and its application is simple and straightforward.

The effectiveness of adaptive fitness sharing was supported by extensive simulation results, and two population diversity measures were developed in order to quantify the obtained results. It was experimentally shown that adaptive fitness sharing, and all its combinations with the technique of population ranking, consistently outperformed the simple GA and population ranking alone. The performance and robustness of the proposed method was further investigated by performing a set of statistical tests, where it was shown that adaptive fitness sharing always outperformed all other methods, in terms of both performance (higher degree of achieved uniformity) and robustness (less sensitivity to initial conditions). Finally, the adaptive properties of the proposed method were demonstrated by modifying the search landscape during the course of a single GA run. It was clearly shown that adaptive fitness sharing successfully adapted the density of the population as required, while maintaining a high degree of uniformity throughout the search run.

5 Decentralised PI Controller Tuning for Multivariable Processes – A Genetic Approach

5.1 Introduction

In this chapter, a new method for the automatic tuning of decentralised PI controllers for multivariable processes is proposed, based on genetic algorithms. The major advantage of the proposed method is that it gives the designer the freedom to explicitly specify the required performance specifications for a given multivariable control problem, in terms of time-domain bounds on the closed-loop responses. This is achieved by transforming the control problem into a function optimisation problem, using objective function J_M developed in Chapter 3. A genetic algorithm is then employed for the minimisation of J_M, and the method of adaptive fitness sharing developed in Chapter 4 is used, in order to maximise the diversity of the obtained family of solutions. The proposed method has the flexibility to be applicable to a wide range of multivariable processes. Simulation results are presented to illustrate the effectiveness of the proposed method. The obtained results are shown to be superior than those obtained using the relay feedback technique. The choice of genetic algorithms as a suitable optimisation method is supported by statistically comparing them with two conventional optimisation methods.

5.2 Limitations of Existing PI/PID Tuning Methods

Although methods exist for the automatic tuning of PI and PID controllers for certain classes of multivariable processes (Åström and Hägglund, 1995; Loh, Tan, and Vasnani, 1994; Halevi, Palmor, and Efrati, 1997; Palmor, Halevi, and Krasney, 1995; Semino and Scali, 1998; Zhuang and Atherton, 1994; Hang, Loh, and Vasnani, 1994), many of these methods make certain assumptions about the nature of the controlled process, such as

size (number or input/output pairs), linearity, weak interactions within the process, absence of noise, and others. A literature search did not reveal a generic, multi-loop PI or PID tuning method that enables a range of different and arbitrary specifications for each loop and between loops, to be defined in the context of an interacting multivariable process. If the controlled process is linear and a simple mathematical model exists or can be derived easily, then it may be possible to derive analytical methods for the tuning of the controllers. However, in the real world, processes are usually non-linear and very complex, and the resulting models are often too complex to be useful in an analytical framework.

A number of successful PI and PID tuning methods for multivariable processes have been proposed by Loh, Tan, and Vasnani (1994), Halevi, Palmor, and Efrati (1997), and Semino and Scali (1998), in which the relay feedback technique developed by Åström and Hägglund (1984) is employed. These methods appear to work well, but have the disadvantage that not all classes of multivariable processes can exhibit sustained and near-sinusoidal oscillations under multi-loop relay feedback. Therefore, these methods are only applicable to certain classes of multivariable processes. Furthermore, in Loh, Tan, and Vasnani (1994), the relay switching levels have to be modified manually, in order to bring the process to a certain mode of oscillations that is necessary for the method to be successfully applied. As the size of the multivariable process increases, this task may become extremely difficult. Another important limitation, as with most existing multi-loop PI/PID tuning methods, is that they employ tuning rules such as those proposed by Ziegler and Nichols (1942), which were originally developed for use with SISO systems and correspond to a fixed set of performance specifications. It will be shown in the following sections that the proposed PI tuning method inherently and completely overcomes these limitations.

5.3 The Proposed Decentralised PI Controller Tuning Method

In this section, a new method is proposed for the tuning of decentralised, multi-loop PI controllers. The tuning problem is transformed into a function optimisation problem by means of objective function J_M developed in Chapter 3, with J_M having $2q$ parameters, where q is the number of PI loops in the closed-loop, multivariable system. The highly non-linear and multimodal nature of J_M, and the lack of derivatives, mainly due to noise

and other uncertain elements that may be present in the closed-loop system, motivates the use of GAs in this optimisation problem. The ability of J_M to deliver a family of strictly optimal solutions is exploited, by using the method of adaptive fitness sharing developed in Chapter 4. The resulting PI tuning method, with the numerical robustness and global optimisation ability of GAs, is expected to overcome limitations of empirical methods based on tuning rules such as those proposed by Ziegler and Nichols (1942). A major advantage of the proposed GA-based tuning method is that the optimality criteria can be explicitly and accurately specified by the designer in the time domain, in terms of the desired transient responses of all closed-loop system outputs under different, user-defined set point patterns, including loop coupling specifications. This makes the method directly applicable to many complex multivariable control problems. Furthermore, although only decentralised PI controllers are considered in this work, the generality and open architecture of the proposed method makes it suitable for the automatic tuning of different parametric controllers, both linear and non-linear, and not just PI controllers. This is because the proposed method only requires the *numerical* solutions of the differential equations associated with the closed-loop system, which can easily be obtained using most standard control system simulation packages.

5.3.1 Decentralised PI Control of Multivariable Processes

In a $q \times q$ multivariable process, a typical decentralised PI controller structure would be one in which q PI controllers would be used in the q loops associated with the process, as shown in Fig. 5.1.

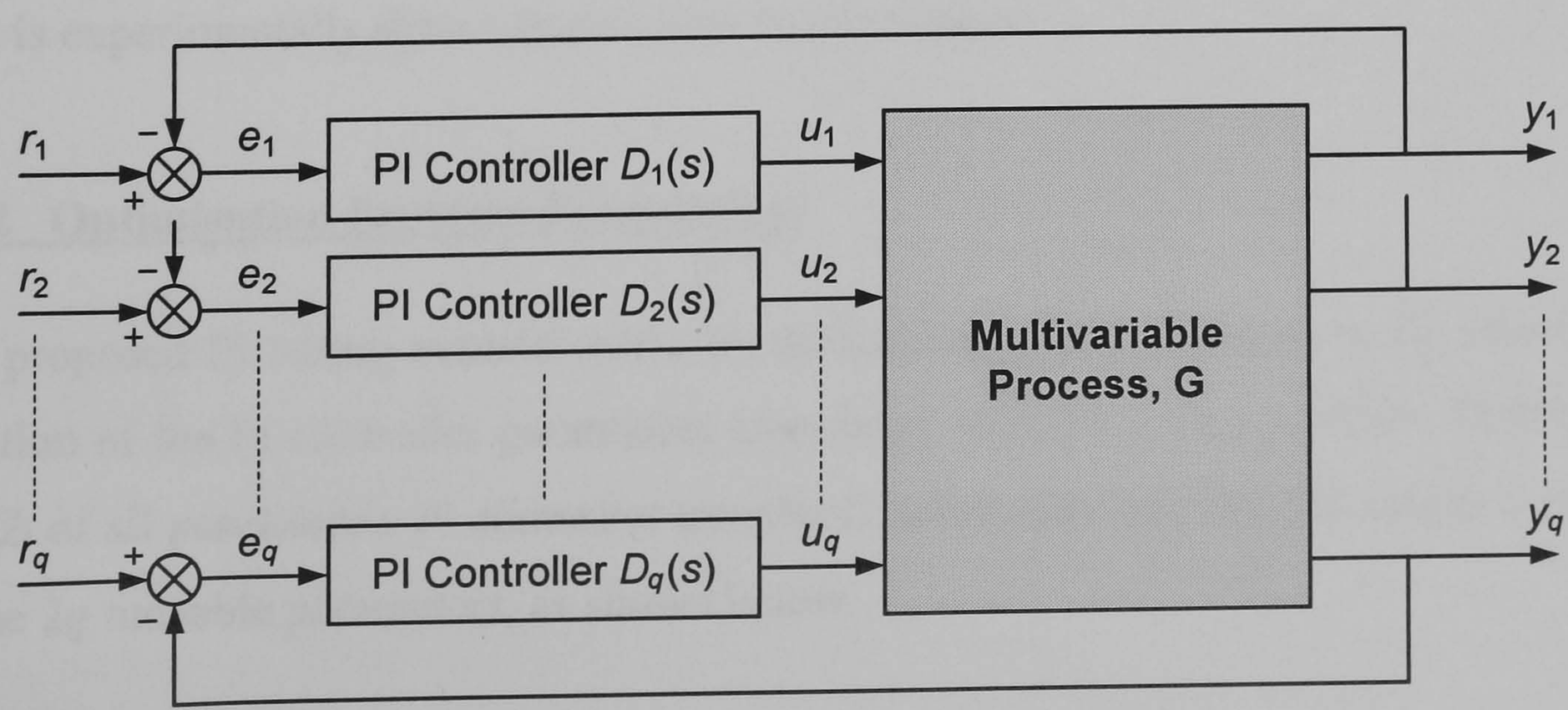


Fig. 5.1 *Typical decentralised PI controller structure*

The pairing of controlled variables y_i with manipulated variables u_i can be performed by examining the process's relative gain array, either in the steady state (Bristol, 1966), or within the desired closed-loop bandwidth. The PI controllers considered in this work are continuous-time, arranged in the standard decentralised structure shown below.

$$\mathbf{D}(s) = \begin{bmatrix} D_1(s) & 0 & \cdots & 0 \\ 0 & D_2(s) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D_q(s) \end{bmatrix} \tag{5.1}$$

The elements in the diagonal of controller transfer function matrix $\mathbf{D}(s)$ are single-input, single-output PI controllers of the following standard form.

$$D_i(s) = K_{P_i}\left(1+\frac{1}{T_{I_i}s}\right), \quad i = 1,\ldots,q \tag{5.2}$$

where K_{P_i} and T_{I_i} denote the proportional gain and integral time, respectively, of the PI controller in loop i. The interactions within the process cause the output of the controller in loop i to appear as a disturbance in all other loops. Hence, the i-th PI controller must be designed such that the desired set point tracking performance for loop i is achieved, while the disturbances caused by the PI controller outputs of the remaining $q-1$ loops are rejected. This makes the PI tuning problem difficult, and conventional PI tuning rules such as those proposed by Ziegler and Nichols (1942) which have been designed for SISO systems may not, in general, achieve acceptable results. The proposed tuning method, however, is inherently capable of treating interacting multivariable processes. This is experimentally demonstrated later in this chapter.

5.3.2 Optimisation Problem Formulation

The proposed PI tuning method works by minimising objective function J_M, which is a function of the PI controller parameters associated with the tuning problem. Hence, the set $\mathcal{D}$ of all permissible PI controller transfer function matrices $\mathbf{D}(s)$, is simply a vector of the $2q$ tuneable parameters, as shown below.

$$\mathcal{D} = \left\{ \left[K_{P_1} \cdots K_{P_q} \;\; T_{I_1} \cdots T_{I_q} \right] \in \mathbb{R}^{2q} \right\} \tag{5.3}$$

In this context, objective function J_M can be used in its original form, and can thus be expressed as shown below.

$$J_M\left(K_{P_1},\ldots,K_{P_q},T_{I_1},\ldots,T_{I_q}\right)=\max_{j=1,\ldots,q}\left\{\sum_{i=1}^{q}w_{ij}J_{ij}\right\} \tag{5.4}$$

where

$$\begin{aligned}J_{ij}\left(K_{P_1},\ldots,K_{P_q},T_{I_1},\ldots,T_{I_q}\right)=\int_0^{t_{max}}&\Big(\max\left\{f_{ij}^{(l)}(t)-y_i(t),0\right\}\\&+\max\left\{y_i(t)-f_{ij}^{(u)}(t),0\right\}\Big)\,dt\end{aligned} \tag{5.5}$$

It is observed that only the process outputs $y_1,\ldots,y_q$ are considered in the minimisation. In certain cases, it may be desirable or necessary to also consider the process inputs (the controller outputs) $u_1,\ldots,u_q$, as well as their rates of change $\dot{u}_1,\ldots,\dot{u}_q$. This is common practice in objective functions employed in predictive control algorithms (García, Prett, and Morari, 1989; Clarke and Mohtadi, 1989), where control weights are used to limit the activity of the manipulated variables. This can also be incorporated in the proposed method as an additional term in J_{ij} or J_M. Depending on how this is done, the inclusion of additional terms in the objective function may alter certain desirable properties of J_M, such as the existence of strictly optimal solutions. This is discussed in Chapter 6. In the present chapter, J_M and J_{ij} are used exactly as shown in (5.4) and (5.5), respectively.

5.3.3 Boundary Functions and Set Point Test Patterns

The set point test patterns used to test the candidate controllers in order to evaluate J_M, the corresponding boundary functions $f_{ij}^{(u)}(t)$ and $f_{ij}^{(l)}(t)$, and the weighting factors w_{ij}, are problem-dependent and must be chosen in accordance with the given specifications. In cases where step functions are used to evaluate the candidate controller, the boundary functions $f_{ij}^{(u)}(t)$ and $f_{ij}^{(l)}(t)$ shown in Fig. 3.17 can be used, by appropriately choosing the constants $c_1,\ldots,c_7$, c_{ss}, $t_1,\ldots,t_3$, and t_{max}. Recall that J_{ij} is the objective function element for output i under set point pattern j. Since the closed-loop system consists of q loops, q set point patterns can be applied, where in pattern j a step function is applied to reference input r_j, while the remaining $q-1$ reference inputs remain at zero. When all q set point patterns have been applied to the closed-loop system, all q^2 elements J_{ij} can be evaluated in order to compute J_M. This is illustrated in Fig. 5.2 for a 2×2 system.

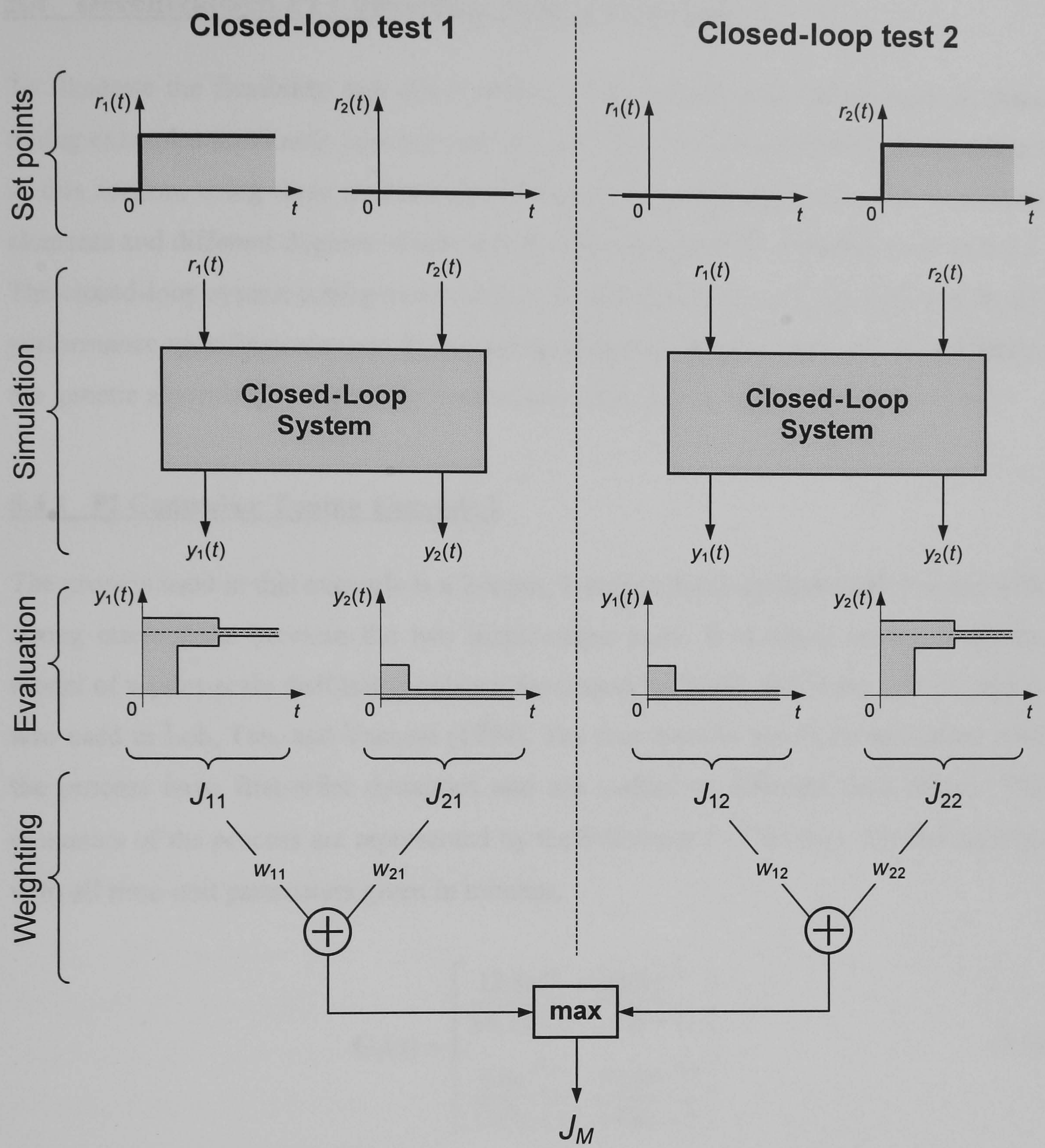


Fig. 5.2 Typical procedure for the computation of J_M for a two-input, two-output system

The procedure shown in Fig. 5.2 is only a suggestion for problems with typical set-point tracking and loop coupling performance specifications. The number of closed-loop tests can be greater than q, especially in cases where the controlled process is non-linear and many different operating points must be considered in the evaluation of the candidate controllers. Furthermore, the set point signals need not be step functions, but can also be ramps or any other suitable function, and can be different for each reference input and closed-loop test combination, provided $f_{ij}^{(u)}(t)$ and $f_{ij}^{(l)}(t)$ are chosen accordingly.

5.4 Decentralised PI Controller Tuning Examples

To illustrate the flexibility and effectiveness of the proposed PI tuning method, three tuning examples previously investigated by Loh, Tan, and Vasnani (1994) are presented in this section, using three multivariable (2-input, 2-output) processes with non-linear elements and different degrees of interaction. These are given in Sections 5.4.1 to 5.4.3. The closed-loop system configuration is then described in Section 5.4.4, followed by the performance specifications used in the tuning, which are given in Section 5.4.5. Finally, the genetic algorithm configuration used in the optimisation is given in Section 5.4.6.

5.4.1 PI Controller Tuning Example 1

The process used in this example is a 2-input, 2-output linear multivariable process with strong interactions between the two input/output pairs. It is based on the empirical model of a pilot-scale distillation column developed by Wood and Berry (1973), and is also used in Loh, Tan, and Vasnani (1994). The four transfer functions associated with the process have first-order dynamics and are subject to different time delays. The dynamics of the process are represented by the following 2×2 transfer function matrix, with all time-unit parameters given in minutes.

$$\mathbf{G}_1(s) = \begin{bmatrix} \dfrac{12.8e^{-s}}{16.7s+1} & \dfrac{18.9e^{-3s}}{21s+1} \\ \dfrac{6.6e^{-7s}}{10.9s+1} & \dfrac{19.4e^{-3s}}{14.4s+1} \end{bmatrix} \tag{5.6}$$

It can be seen by examining the off-diagonal elements of $\mathbf{G}_1(s)$ that the interactions within the process are strong, and the delay times range from 1 minute to 7 minutes. It will be shown later in this chapter that even simple MIMO processes such as $\mathbf{G}_1(s)$ can generate very complex search landscapes containing thousands of local minima.

5.4.2 PI Controller Tuning Example 2

In this example, $\mathbf{G}_1(s)$ is modified in order to reduce the interactions within the process. Specifically, the steady-state gains of the off-diagonal elements of $\mathbf{G}_1(s)$ are reduced, resulting in the following transfer function matrix.

$$\mathbf{G}_2(s) = \begin{bmatrix} \dfrac{12.8e^{-s}}{16.7s+1} & \dfrac{0.2e^{-3s}}{21s+1} \\ \dfrac{e^{-7s}}{10.9s+1} & \dfrac{19.4e^{-3s}}{14.4s+1} \end{bmatrix} \tag{5.7}$$

5.4.3 PI Controller Tuning Example 3

Finally, in this example, the gain of the $G_{12}(s)$ element of $\mathbf{G}_1(s)$ is reduced, in order to weaken the interaction from loop 2 to loop 1, but the interaction from loop 1 to loop 2 is left unchanged, thus resulting in a 'moderate' degree of interaction. This results in the following transfer function matrix.

$$\mathbf{G}_3(s) = \begin{bmatrix} \dfrac{12.8e^{-s}}{16.7s+1} & \dfrac{10e^{-3s}}{21s+1} \\ \dfrac{6.6e^{-7s}}{10.9s+1} & \dfrac{19.4e^{-3s}}{14.4s+1} \end{bmatrix} \tag{5.8}$$

5.4.4 Closed-Loop System Formulation

In all three tuning examples, y_1 was paired with u_1 and y_2 with u_2 as indicated by the steady-state relative gain arrays of $\mathbf{G}_1(s)$, $\mathbf{G}_2(s)$, and $\mathbf{G}_3(s)$, all of which have positive elements in the main diagonal and negative elements elsewhere. Two continuous-time PI controllers in the standard form of Eq. (5.2) were used to control the multivariable processes. In all three examples, the processes were augmented with saturation (±0.5 units) and rate limit (±0.015 units per minute) non-linearities as actuator constraints. These non-linearities, as well as the existence of time delays in the process, limit the use of conventional, linear systems control theory for the design of the PI controllers, as such designs may not, in general, yield reliable results. The resulting closed-loop system is shown in Fig. 5.3.

In order to evaluate J_M, the closed-loop systems were simulated in MATLAB/SIMULINK using the Runge-Kutta fifth-order numerical integration algorithm with a constant step size of $t_{max}/100$ minutes, where t_{max} is the maximum simulation time in each example. The integral in Eq. (5.5) was numerically evaluated using the Euler method.

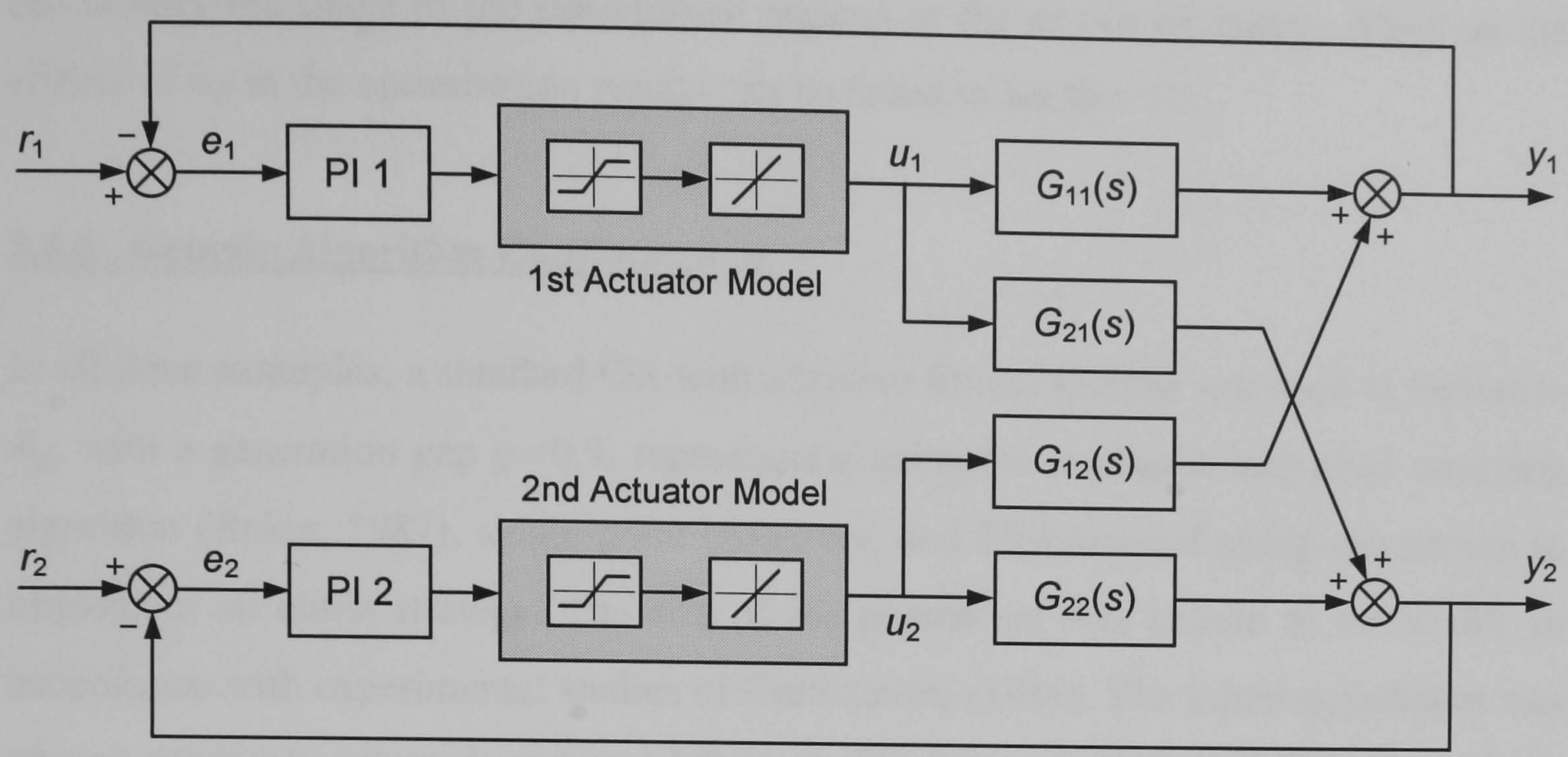


Fig. 5.3 The closed-loop configuration used in the PI controller tuning examples

5.4.5 Performance Specifications

In all examples, the set point tracking and loop coupling performance specifications are of the same form as the ones shown in Fig. 3.17. The closed-loop system was subjected to unit step functions, as shown in Fig. 5.2, and the values of the constants $c_1, \ldots, c_7$, c_{ss}, $t_1, \ldots, t_3$, and t_{max} in Fig. 3.17 were chosen so that the performance specifications of the closed-loop system are as follows.

Peak overshoot of outputs y_1 and y_2:	$\leq 20\%$	
Settling time of outputs y_1 and y_2:	≤ 50 min,	where the settling time is 5%
Coupling between loops 1 and 2:	$\leq 50\%$,	$0 \leq t \leq 25$ min
	$\leq 5\%$,	25 min $< t \leq 100$ min

In all examples, the weighting factors w_{ij} in Eq. (5.4) were set to be $w_{11} = w_{22} = 1$ and $w_{12} = w_{21} = 0.25$, in order to give more emphasis to the set point tracking objectives. As mentioned in Chapter 3, the weighting factors w_{ij} do not at all affect the locations of the strictly optimal solutions in the search landscape. However, w_{ij} should still be chosen according to the importance of the corresponding specifications, since the existence of strictly optimal solutions is not known *a priori*. Furthermore, even when strictly optimal solutions do exist, the values of w_{ij} can affect the convergence of the GA because they

can modify the shape of the sub-optimal regions of the search landscape. More on the effects of w_{ij} in the optimisation results can be found in Section 3.5.

5.4.6 Genetic Algorithm Configuration

In all three examples, a standard GA with adaptive fitness sharing was used to minimise J_M, with a generation gap $g=0.9$, reproduction using the stochastic universal sampling algorithm (Baker, 1987), single-point crossover, and fitness-based string reinsertion to implement an elitist strategy. The size of the population was chosen to be $N=80$, in accordance with experimental studies of Grefenstette (1986). The initial population was always randomly selected, and was left to evolve for a maximum of 100 generations, although good convergence was usually achieved in much fewer generations.

The four PI controller tuning parameters were encoded using the binary alphabet and Gray coding (Caruana and Schaffer, 1988). The string resolution for each parameter was set to 12 bits, resulting in a total string length of $l=4\times12=48$ bits. This corresponds to a search space whose size is approximately 2.8×10^{14} points. The chromosome structure used in all tuning examples is shown in Fig. 5.4 below.

$$\overbrace{\underbrace{01101\ldots10}_{K_{P_1}}\ \underbrace{11100\ldots11}_{T_{I_1}}}^{\text{PI controller in loop 1}}\ \overbrace{\underbrace{00010\ldots00}_{K_{P_2}}\ \underbrace{11101\ldots01}_{T_{I_2}}}^{\text{PI controller in loop 2}}$$

Fig. 5.4 *Chromosome structure used in the tuning examples*

The crossover and mutation probabilities were chosen to be $p_c=0.45$ and $p_m=0.01$, respectively, using the guidelines of Grefenstette (1986). The PI controller proportional terms K_{P_i} were assumed to take values in the interval $[0,10]$, and the integral terms T_{I_i} in the interval $[0.1,100]$ minutes. Adaptive fitness sharing with pre-ranking was used as the fitness assignment strategy in all examples. The sharing radius σ_{share} was computed using Eq. (4.17), where $n=2q=4$, and the value of b was taken from Table 4.1.

$$\sigma_{share} \approx \frac{\sqrt[n]{b}}{\sqrt[n]{N}-1} = \frac{\sqrt[4]{\frac{1}{32}\pi^2}}{\sqrt[4]{80}-1} = 0.3744 \tag{5.9}$$

5.5 Simulation Results

5.5.1 Tuning Example 1

Simulation results for the best set of PI controller parameters after 100 generations are shown in Fig. 5.5. In the output responses, the boundary functions $f_{ij}^{(u)}(t)$ and $f_{ij}^{(l)}(t)$ are also plotted for comparison, indicated by the dotted lines. The convergence of the value of J_M and that of the proportional and integral terms of the two PI controllers is shown in Fig. 5.6, where the vertical dashed line indicates the generation at which the GA has identified one or more strictly optimal solutions ($J_M=0$).

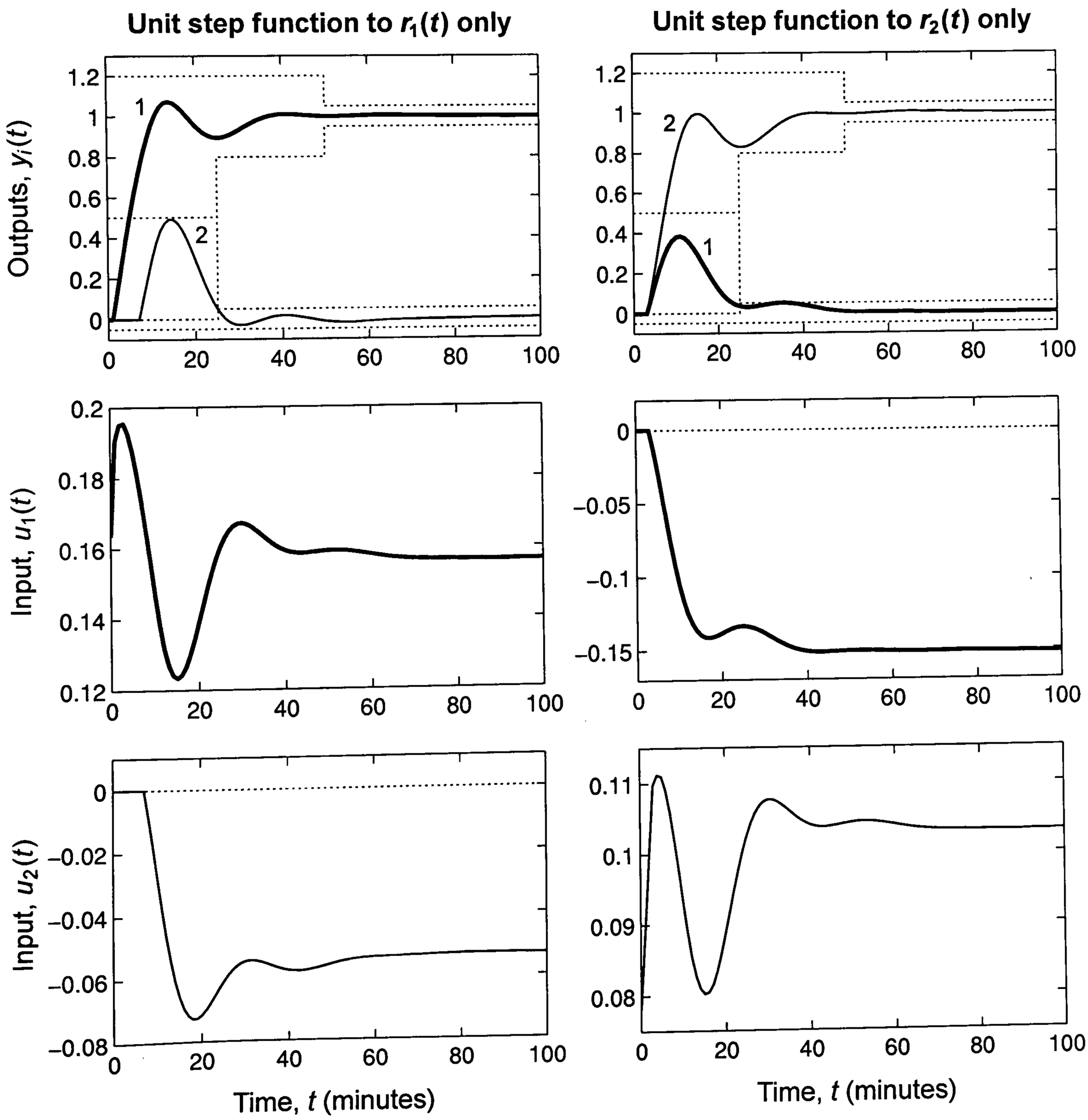


Fig. 5.5 *Strictly optimal closed-loop system responses (PI tuning example 1)*

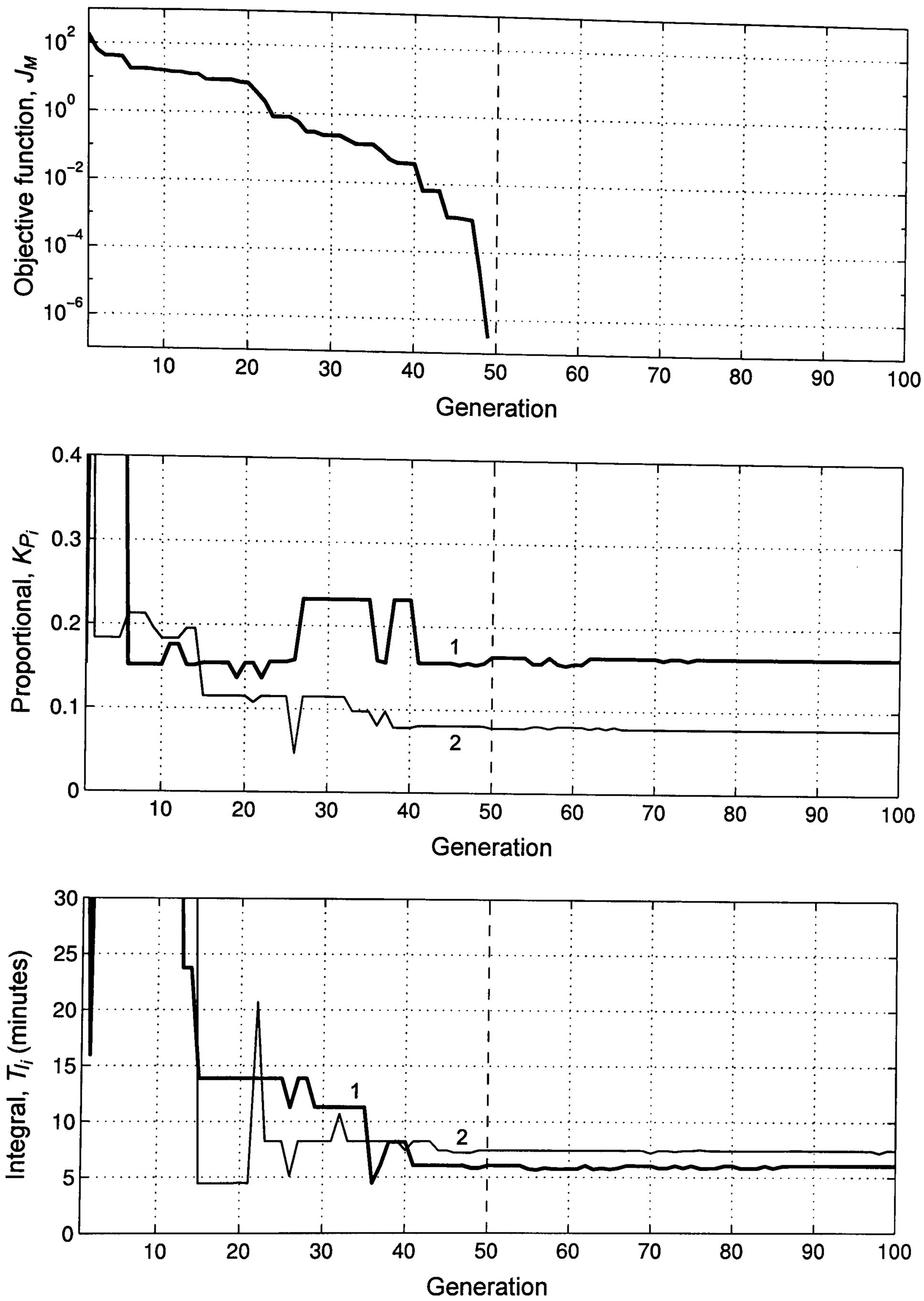


Fig. 5.6 *Convergence of J_M and the four PI controller parameters (PI tuning example 1)*

It can be clearly seen that the closed-loop system has completely met all specifications. Although strictly optimal solutions ($J_M=0$) were obtained as early as generation 50, the population was left to evolve for 100 generations, so that other strictly optimal solutions

are obtained using adaptive fitness sharing. It was observed that the resulting solutions were very similar. This is because the performance specifications for the given process were very tight, resulting in the n-volume of $\tilde{\mathcal{D}}$ being extremely small. Evidence of the small n-volume of $\tilde{\mathcal{D}}$ in this example can be found in Fig. 5.6, where the variance of the controller parameters after generation 50 is low.

5.5.2 Tuning Example 2

Simulation results for the family of strictly optimal PI controllers for $\mathbf{G}_2(s)$ obtained at generation 100 are shown in Fig. 5.7. The GA convergence plots are shown in Fig. 5.8.

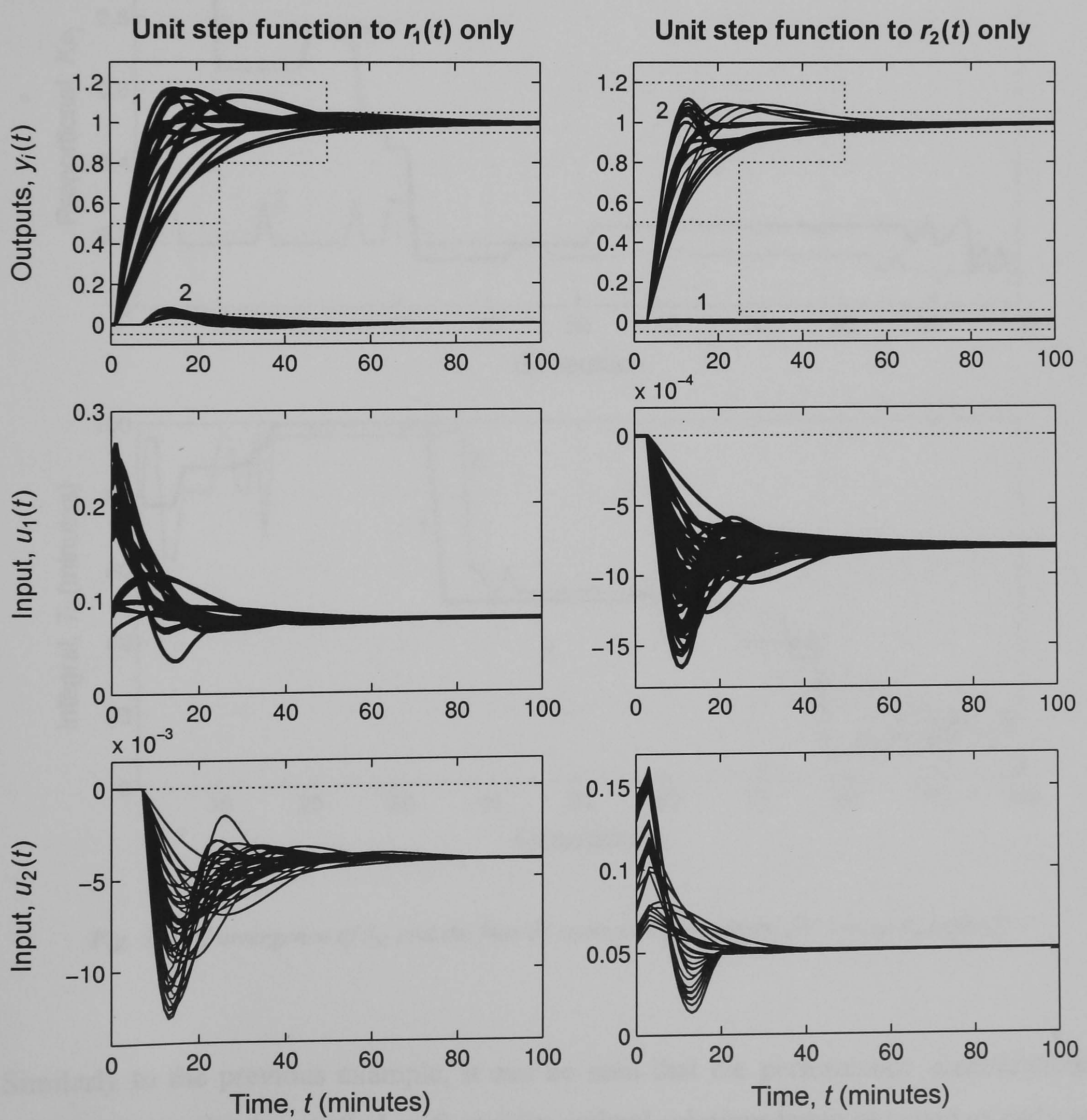


Fig. 5.7 *Family of strictly optimal closed-loop system responses (PI tuning example 2)*

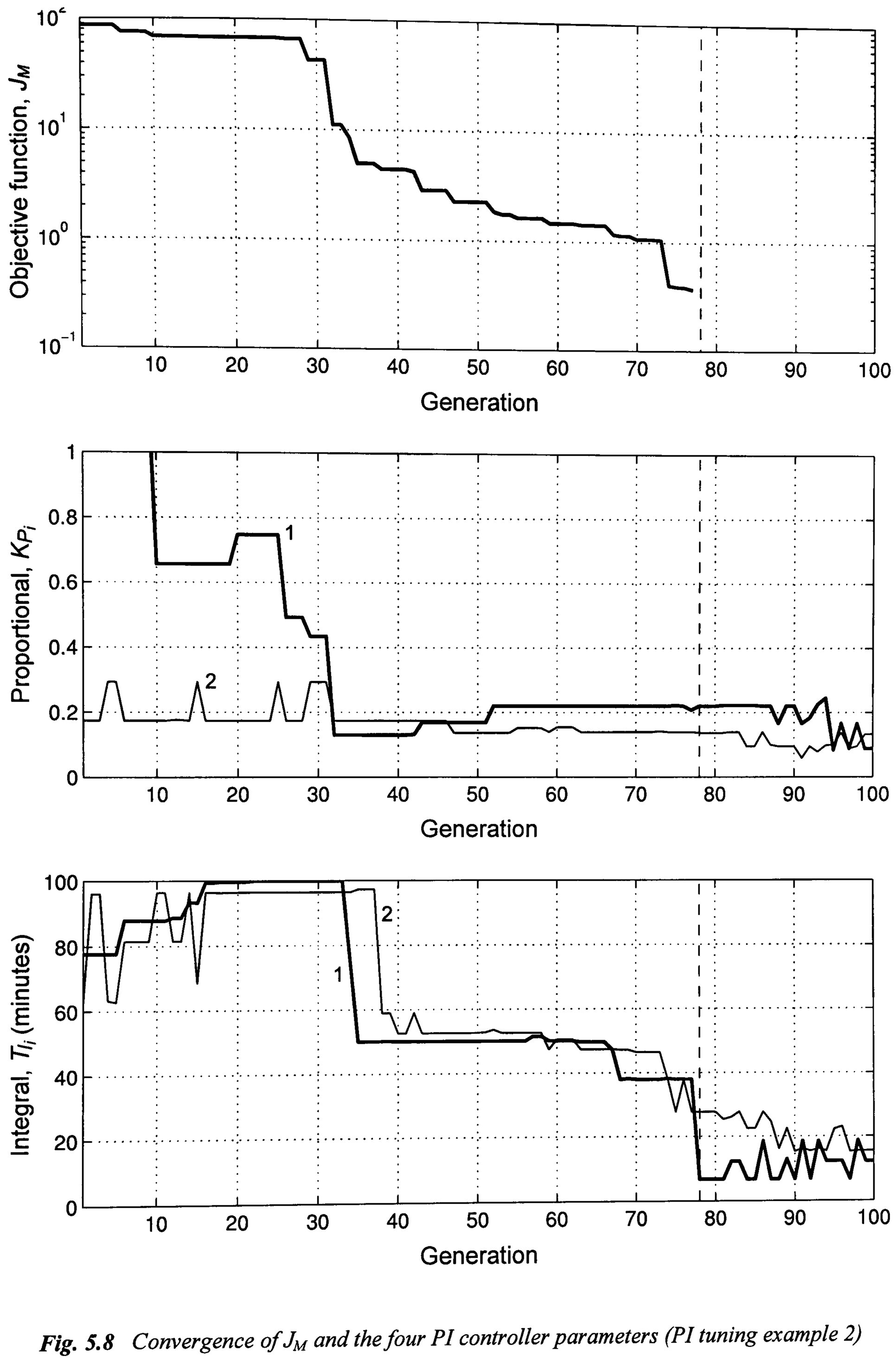


Fig. 5.8 *Convergence of J_M and the four PI controller parameters (PI tuning example 2)*

Similarly to the previous example, it can be seen that the performance specifications have been completely satisfied, with strictly optimal solutions being obtained as early as generation 78. It is observed that in this tuning example, the GA was able to locate

many significantly different strictly optimal solutions. This was expected because the process used in this example is almost diagonal due to the very weak interactions within the process, making the performance specifications realisable by a much wider range of PI controllers. Evidence of the large n-volume of $\tilde{\mathcal{D}}$ in this example can be found in Fig. 5.8, where the variance of the controller parameters after generation 78 is high.

5.5.3 Tuning Example 3

Simulation results for the family of strictly optimal PI controllers for $\mathbf{G}_3(s)$ obtained at generation 100 are shown in Fig. 5.9. The GA convergence plots are shown in Fig. 5.10.

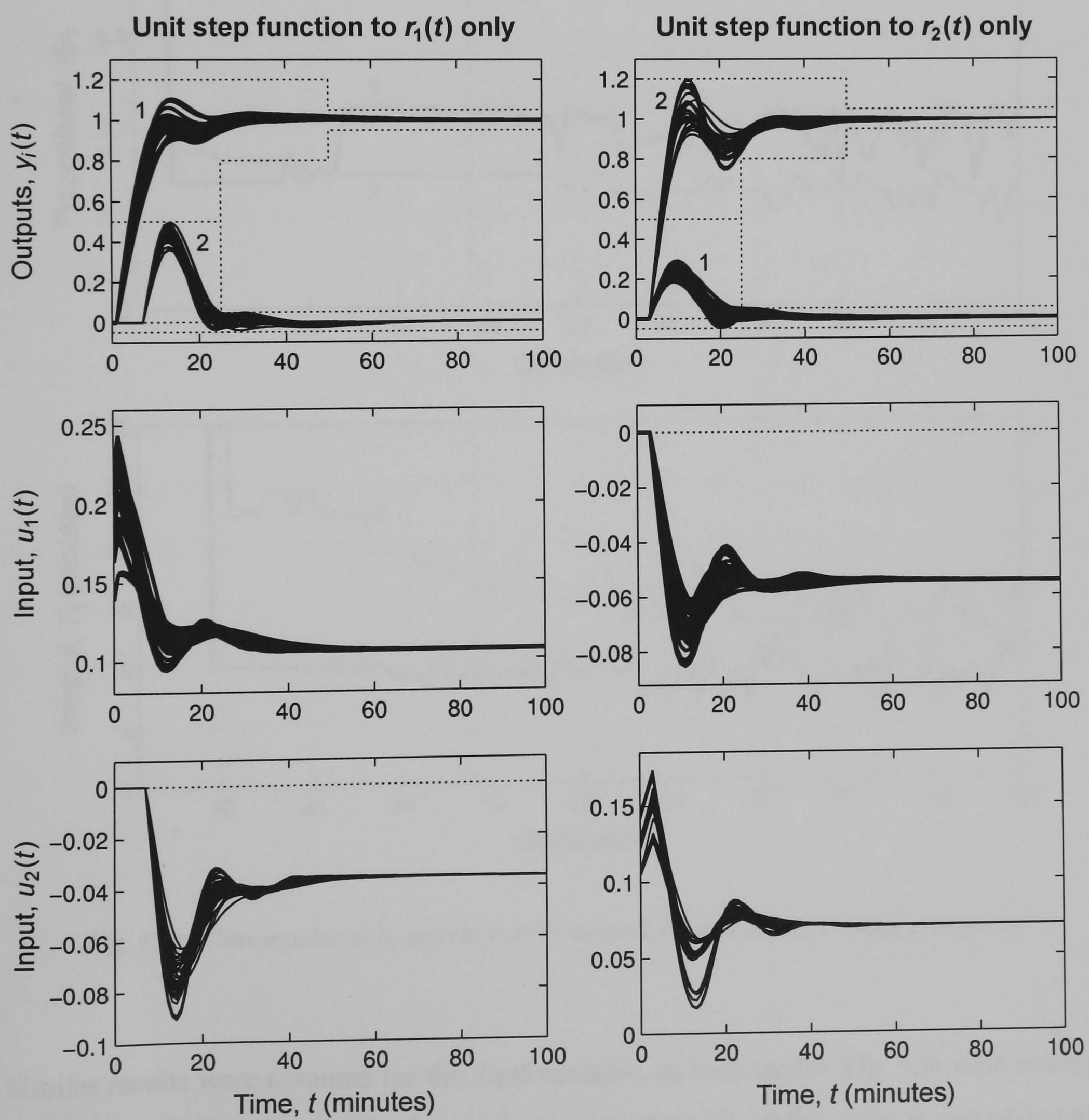


Fig. 5.9 *Family of strictly optimal closed-loop system responses (PI tuning example 3)*

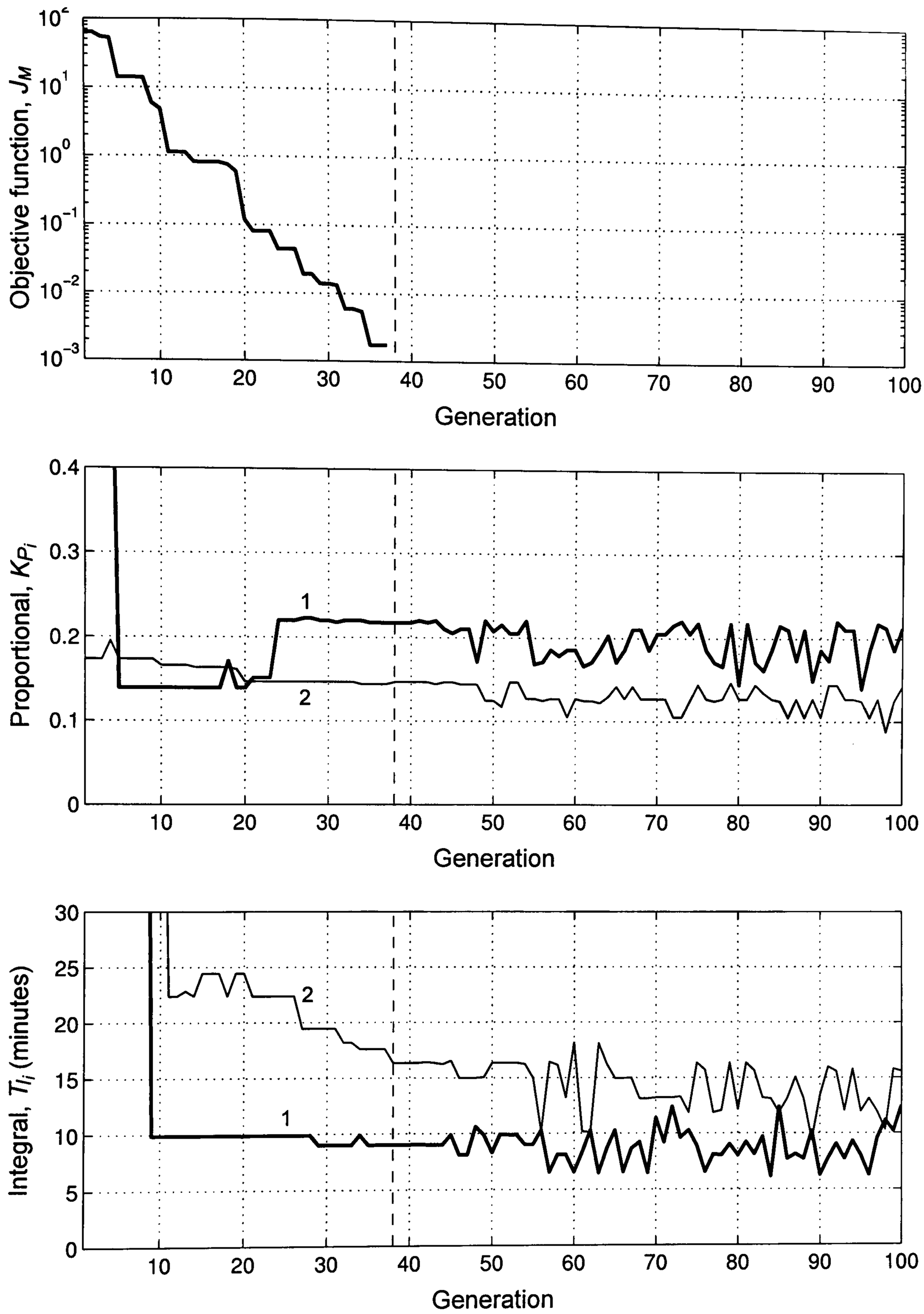


Fig. 5.10 *Convergence of J_M and the four PI controller parameters (PI tuning example 3)*

Similar results were obtained for the third example, as indicated in Fig. 5.9, with strictly optimal solutions being obtained as early as generation 38. In this case, it was observed that the envelope of achievable strictly optimal output responses was narrower than that

of example 2. This was expected, since $\mathbf{G}_3(s)$ exhibits stronger interactions than $\mathbf{G}_2(s)$. Evidence of the large n-volume of $\tilde{\mathfrak{D}}$ in this example can be found in Fig. 5.10, where the variance of the controller parameters after generation 38 is high.

The high variance observed in the tuning parameter values in examples 2 and 3 towards the end of the GA search run should not be treated as an indication of poor convergence. The best solutions in and after the generation marked by the dashed line in Figures 5.6, 5.8 and 5.10 are all strictly optimal, and the tuning parameters should be expected to stay in $\tilde{\mathfrak{D}}$, but not necessarily settle at specific values. Another relevant observation is that in examples 2 and 3, the variance of the tuning parameters appears to increase after the GA has located strictly optimal solutions. This is because prior to the discovery of solutions in $\tilde{\mathfrak{D}}$, the entire population contains sub-optimal solutions. Therefore, the best solution in the population at each generation (plotted in Figures 5.6, 5.8 and 5.10) is likely to be unique among the distinct members of the population. Since $g<1$ and the string reinsertion is fitness-based, this solution will be propagated through to successive generations until a better solution is discovered. When the GA has run long enough to have located the optimal region in the search landscape, this new solution is likely to be in the phenotypic neighbourhood of its predecessor. This similarity between successive best solutions reduces the parameter variance prior to the discovery of solutions in $\tilde{\mathfrak{D}}$. When at least two distinct solutions in $\tilde{\mathfrak{D}}$ are discovered, adaptive fitness sharing is activated in order to maximise the solution diversity in $\tilde{\mathfrak{D}}$. This diversity maximisation is directly responsible for the increase in the variance of the parameters after the GA has located $\tilde{\mathfrak{D}}$, as clearly observed in Figures 5.8 and 5.10.

The resulting strictly optimal PI controller parameters for the three tuning examples, as well as the associated objective function values, are shown in Table 5.1 below.

TABLE 5.1 Typical, strictly optimal PI controller parameters for the three PI tuning examples

	Tuning example 1		**Tuning example 2**		**Tuning example 3**	
	Loop 1	**Loop 2**	**Loop 1**	**Loop 2**	**Loop 1**	**Loop 2**
Proportional, K_{P_i}	0.1636	0.0781	0.1832	0.1294	0.2173	0.1270
Integral, T_{I_i} (minutes)	6.0525	7.4187	6.3941	15.3960	8.8336	16.3719
Objective function, J_M	0 (strictly optimal)		0 (strictly optimal)		0 (strictly optimal)	

The PI controller parameter values shown in Table 5.1 for examples 2 and 3 are typical representatives of the solution families obtained by the GA, and were selected manually by examining the strictly optimal closed-loop responses.

5.5.4 Search Landscape Complexity

The search landscapes that correspond to the three PI tuning examples presented earlier are all four-dimensional and thus cannot be visualised easily. It is possible, however, to examine the surface slices that result by keeping any two of the four parameters at their optimal values given in Table 5.1. One such surface slice for tuning example 1 is shown in Fig. 5.11, obtained by keeping the two proportional terms at their optimal values and varying the two integral terms.

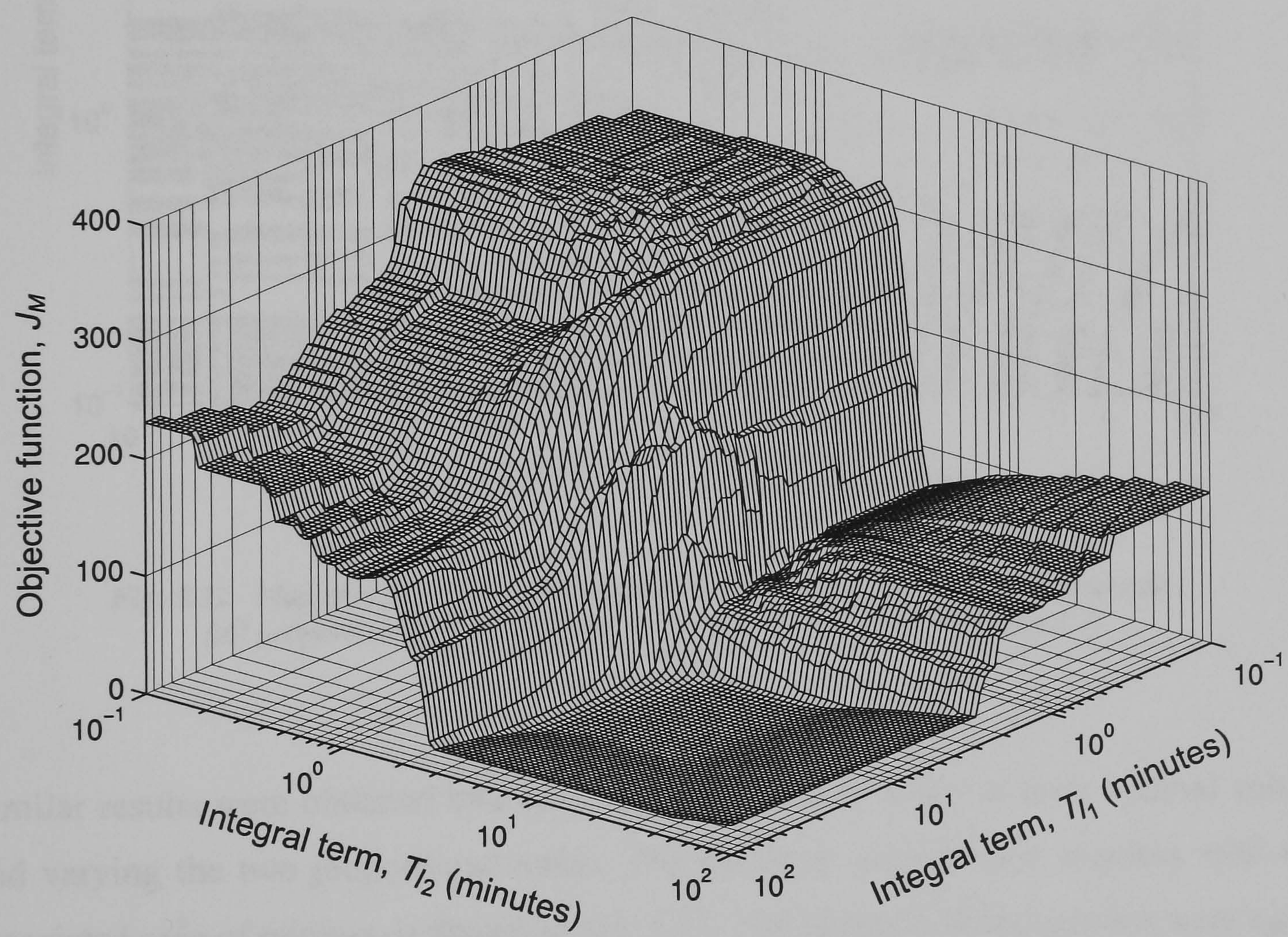


Fig. 5.11 *Objective function surface slice for PI tuning example 1 (all proportional terms are at their optimal values shown in Table 5.1)*

It is observed that the surface slice is complex, highly non-linear and also multimodal. The multimodality of the surface slice can better be observed in Fig. 5.12, where the map of the minima of that surface slice are shown. The locations of the minima were

computed by evaluating J_M at a grid of 401×401 points where 8,630 minima were found, of which only 14 (≈0.16%) correspond to strictly optimal solutions (J_M=0). These are indicated by the circular marker in Fig. 5.12.

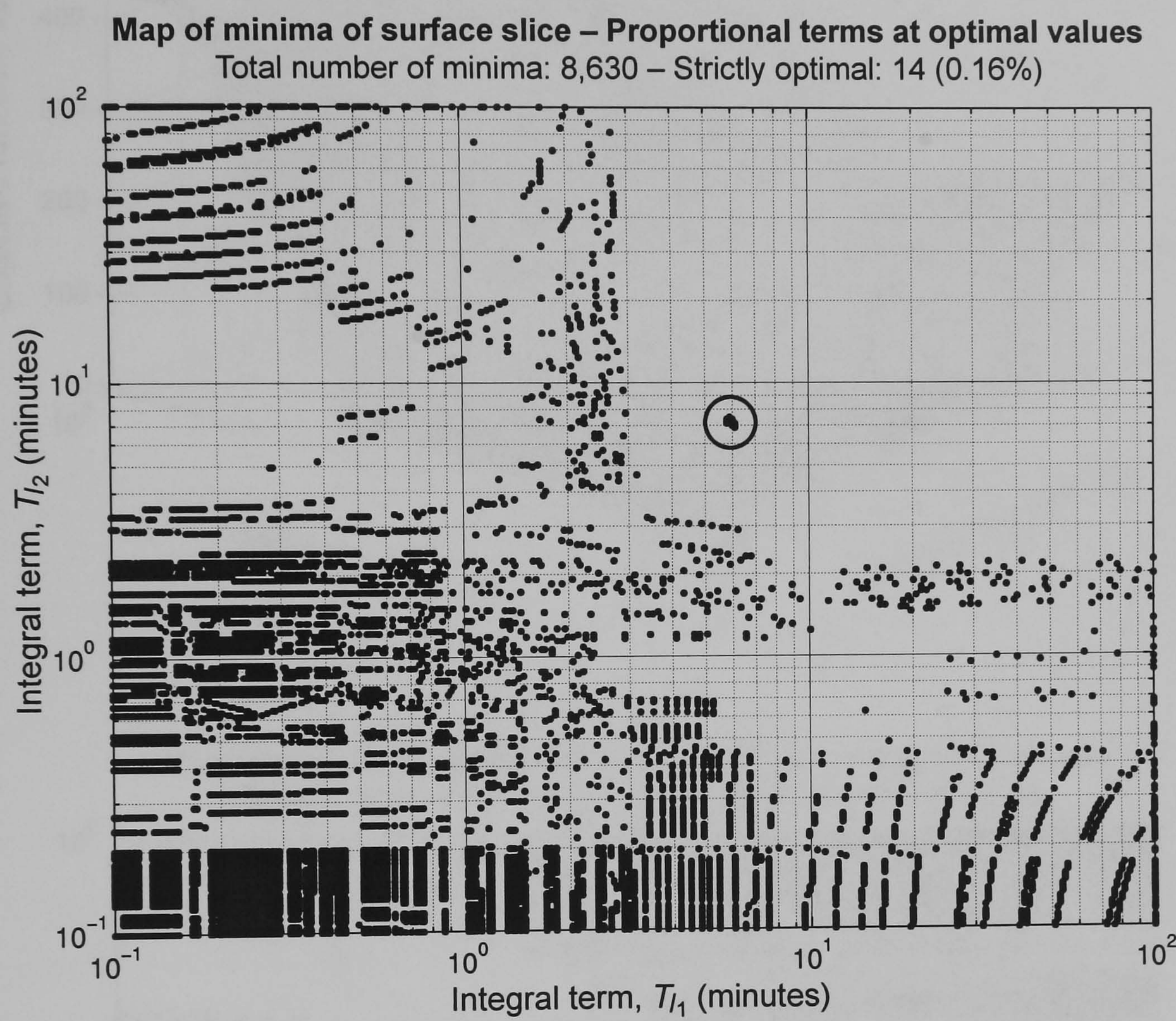
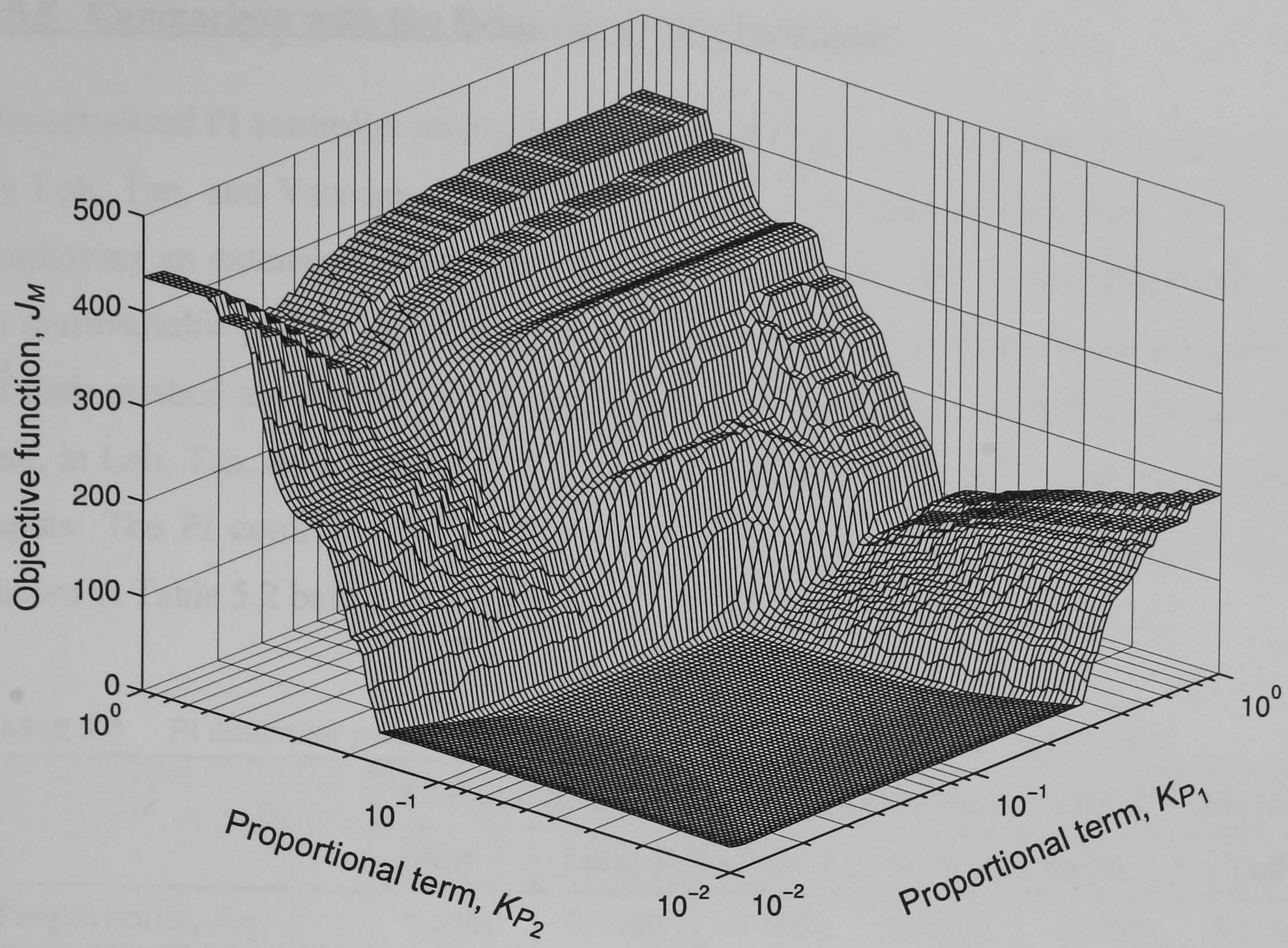


Fig. 5.12 *Map of minima of objective function surface slice for PI tuning example 1 (all proportional terms are at their optimal values shown in Table 5.1)*

Similar results were obtained by keeping the two integral terms at their optimal values and varying the two proportional terms. The resulting surface slice, together with the associated map of minima, is shown in Fig. 5.13. The locations of the minima were again computed by evaluating J_M at a grid of 401×401 points, where 9,453 minima were found, of which only 2 (≈0.02%) correspond to strictly optimal solutions (J_M=0). These are indicated by the circular marker in Fig. 5.13. Note that, in both cases, the minima were computed by assuming that J_M changes monotonically between grid points. This should generally hold if the grid is sufficiently dense. The actual 'continuous' surface slices may contain a much larger number of minima.



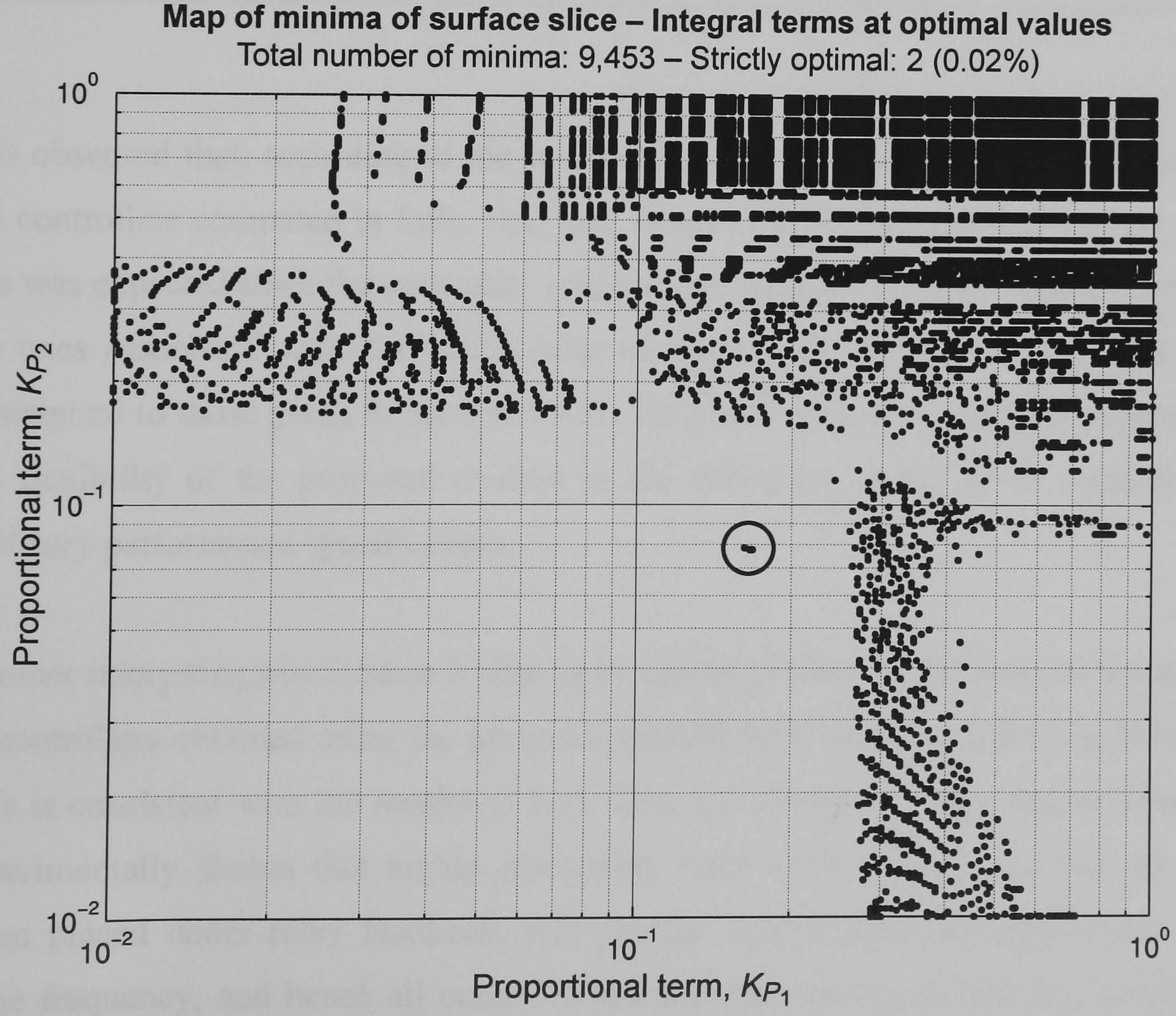


Fig. 5.13 *Objective function surface slice and map of minima for PI tuning example 1 (all integral terms are at their optimal values shown in Table 5.1)*

5.5.5 Comparison with the Relay Feedback Technique

Decentralised PI controller tuning for multivariable processes has also been investigated by Loh, Tan, and Vasnani (1994), who proposed a solution to the tuning problem by employing an extension of the relay feedback technique (Åström and Hägglund, 1984) to multivariable plants. The tuning examples they used to demonstrate the effectiveness of their method are identical to those presented in this chapter, with the only exception that, in Loh, Tan, and Vasnani (1994), no actuator constraints are present in the process inputs. The PI controller parameters they obtained, as well as the values of J_M, are shown in Table 5.2 below.

TABLE 5.2 PI controller parameters as computed in Loh, Tan, and Vasnani (1994)

	Tuning example 1		Tuning example 2		Tuning example 3	
	Loop 1	Loop 2	Loop 1	Loop 2	Loop 1	Loop 2
Proportional, K_{P_i}	0.3160	0.1080	0.7580	0.1540	0.4390	0.1160
Integral, T_{I_i} (minutes)	10.2200	10.2200	3.2400	8.6000	9.6000	9.6000
Objective function, J_M	0.1042		0.7439		0.2327	

It is observed that, according to the performance specifications given in Section 5.4.5, the controllers computed in Loh, Tan, and Vasnani (1994) are sub-optimal. Of course, this was expected since the optimality criteria used in Loh, Tan, and Vasnani (1994) are the ones associated with the tuning rules of Ziegler and Nichols (1942), and do not correspond to those given in Section 5.4.5. The above comparison merely demonstrates the flexibility of the proposed method in the automatic tuning of PI controllers for *arbitrary* performance specifications.

Another interesting observation is that, in PI tuning example 1, the integral terms of the PI controllers obtained using the proposed method have similar values (see Table 5.1). This is consistent with the results of Loh, Tan, and Vasnani (1994), where it has been experimentally shown that highly interacting multivariable processes such as $\mathbf{G}_1(s)$, when placed under relay feedback, will usually exhibit sustained oscillations of the same frequency, and hence all control loops will have the same ultimate period, thus resulting in PI controllers that have the same integral term setting. This can be observed in Table 5.2.

The systems used in the three tuning examples are considered non-linear because of the saturation and rate limit non-linearities that are present in the process inputs. Therefore, as expected, the controller parameters in Table 5.1 are guaranteed to be strictly optimal only when the systems are subjected to the same input patterns as the ones used in the GA for the optimisation (i.e. unit step functions), and may not be strictly optimal when the systems are subjected to step functions with magnitudes other than unity. This is due to the fact that the PI controllers are linear and thus cannot be optimal over the entire process operating region. In the extension of the relay feedback method to multivariable processes, an infinite number of sets of ultimate quantities can be obtained by varying the loop relay switching levels, thus yielding an infinite number of controllers, and this is true even for linear processes (Halevi, Palmor, and Efrati, 1997; Palmor, Halevi, and Krasney, 1995). The proper adjustments of the loop relay switching levels, so that the resulting ultimate quantities reflect the operating point of interest, can be a very difficult task that is largely based on trial-and-error. This is especially true when, at the same time, the process has to exhibit a certain mode of oscillations necessary for the relay feedback method to be successfully applied. More details on the different modes of oscillations that may arise in a multivariable process under relay feedback can be found in Loh and Vasnani (1994), who showed that the obtained limit cycles are related to the strength of the interactions within the controlled process. Processes $\mathbf{G}_1(s)$, $\mathbf{G}_2(s)$, and $\mathbf{G}_3(s)$, used in the tuning examples in this chapter, exhibit all three modes of oscillations investigated in Loh and Vasnani (1994), and Loh, Tan, and Vasnani (1994).

In the proposed PI tuning method, the operating points of interest are embedded in the performance specifications. Therefore, the tuning task is greatly simplified. In addition to that, the PI controllers can be made robust over different operating conditions by defining appropriate set point test patterns that cover all operating points of interest and specifying the corresponding performance specifications. A limitation of the proposed method is that a suitable process model and the use of a simulator may be required, in order to perform the closed-loop tests necessary to evaluate J_M. This is mainly because of the stochastic nature of GAs which may produce solutions (controllers) that cannot be directly applied to the real process for safety and other reasons. However, a similar limitation exists for the relay feedback technique, in which the real process is forced to exhibit limit-cycle oscillations, something that may not be allowed in certain types of processes for safety and other reasons.

5.6 Comparison with Conventional Optimisation Methods

In order to evaluate the performance of GAs in the PI tuning examples presented earlier, the Sequential Quadratic Programming (SQP) algorithm and a pure random search were used to minimise J_M in PI tuning example 1. The SQP algorithm utilises a quasi-Newton Hessian approximation using the BFGS updating method. An overview of SQP can be found in Fletcher (1987), Gill, Murray, and Wright (1981), and Powell (1983). All tests were performed with and without actuator and measurement noise in the closed-loop system. The actuator and measurement noise signals were normally distributed random sequences, having zero mean and variances $\sigma_a^2 = 2\times10^{-5}$ and $\sigma_m^2 = 5\times10^{-4}$, respectively. Furthermore, additional GA tests were performed, in which the crossover and mutation operators were not applied, to demonstrate their effect in the performance of the GA. In order to obtain statistically significant results, the GA and random search algorithms were run 200 times each, and the SQP algorithm 5000 times. The resulting optimisation success rates for two different convergence criteria are shown in Fig. 5.14 below.

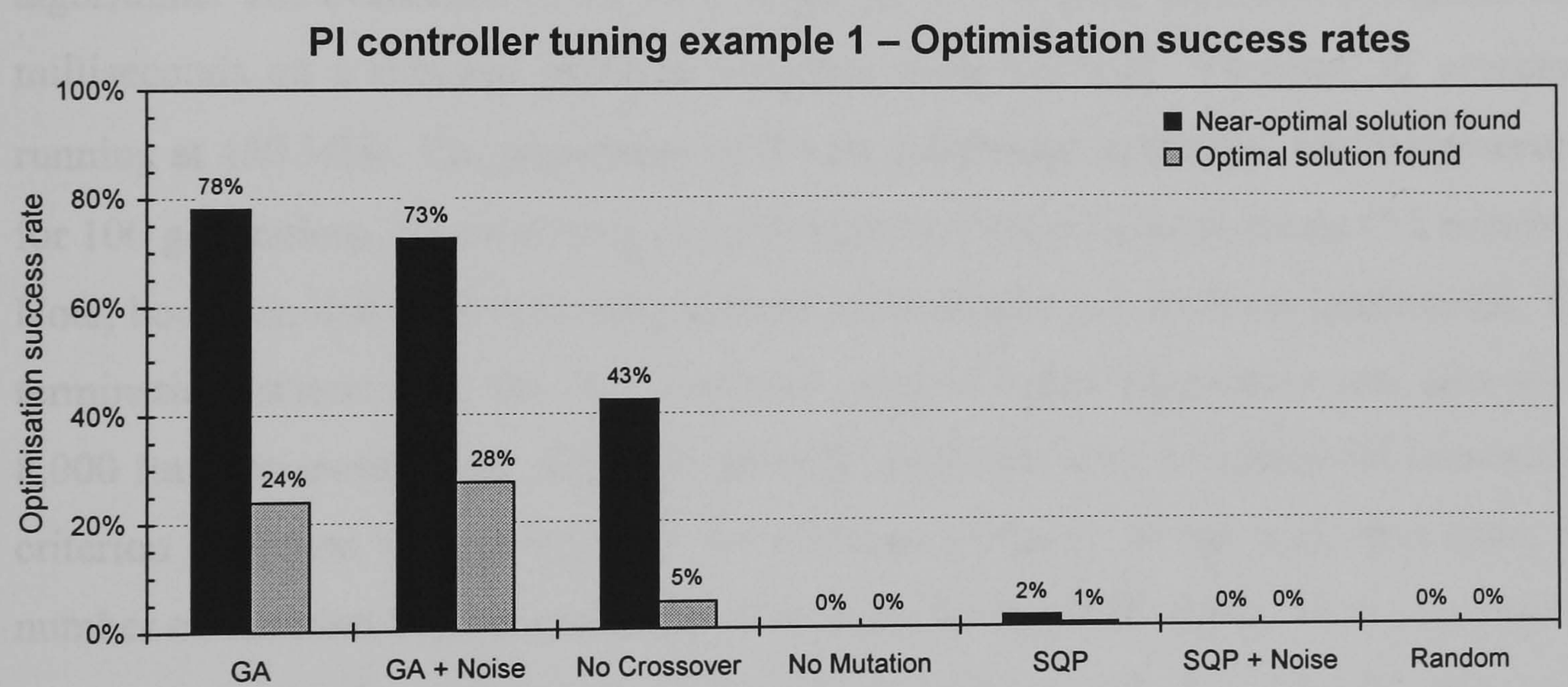


Fig. 5.14 *Optimisation success rates for two convergence criteria (PI tuning example 1)*

It is clearly observed that the performance of the GA is superior to that of both the SQP algorithm and the pure random search. In the case where only near-optimal solutions are required ($J_M \leq 0.05$), it can be seen that the GA is very robust and immune to noise, with success rates between 73% and 78%. On the other hand, the SQP algorithm performed poorly, with less than 2% success in the noise-free case and no successful runs at all

when noise was added to the system. The pure random search was also not successful, even in the noise-free case. When strictly optimal solutions are required ($J_M=0$), it is observed that the performance of the GA is again significantly better than that of the other methods, with success rates between 24% and 28%. Again, the SQP algorithm performed poorly, with less than 1% success in the noise-free case and no successful runs at all when noise was added to the system. The pure random search was again not successful. In the case where no crossover is applied, the ability of the GA to locate strictly optimal solutions was greatly reduced, thus indicating that crossover improves the exploitation ability of the GA. In the case where no mutation is applied, the GA was unable to locate even near-optimal solutions, thus demonstrating the very important role of mutation in introducing new genetic material in the population of the GA.

5.6.1 Computation Time Comparisons

In all three optimisation algorithms, the majority of computation time was taken up by the evaluation of the objective function, rather than the computations involved in the algorithms. The evaluation of J_M for a single set of controller parameters required 38.9 milliseconds on a standard personal computer with the Intel® Pentium® III processor running at 450 MHz. The population of $N=80$ individuals in the GA was left to evolve for 100 generations, thus resulting in 8,000 objective function evaluations (5.2 minutes). Note, however, that both optimality criteria were usually met in fewer generations. The termination criterion for the SQP and pure random search algorithms was also set to 8,000 function evaluations, although the SQP algorithm uses an additional termination criterion based on the precision of the obtained solution. In the noise-free case, the number of function evaluations required in order for the SQP algorithm to successfully reach near-optimal solutions ($J_M \leq 0.05$) varied between 216 (8.4 seconds) and 1,556 (1 minute), with an average of 722 (28.1 seconds). When strictly optimal solutions are required ($J_M=0$), the number of function evaluations required by the SQP algorithm varied between 216 (8.4 seconds) and 1,487 (57.8 seconds), with an average of 595 (23.1 seconds). Although the computation time required by the SQP algorithm appears shorter than the GA case, it should be noted that the SQP algorithm only achieved near-optimal solutions in less than 2% of all test runs. Hence, in order to obtain reliable results, the SQP algorithm requires a large number of runs from different starting points in the search space, which makes the overall computation time comparable to that of the

GA. In the case where noise was added to the system, the SQP algorithm was unable to converge to a minimum. Furthermore, the proposed GA-based PI tuning method has the ability to locate a family of strictly optimal solutions, as opposed to the single solution obtained by the SQP algorithm. The pure random search algorithm is not discussed in this section because it was not successful in obtaining either near-optimal ($J_M \leq 0.05$) or strictly optimal ($J_M = 0$) solutions.

5.7 Summary

In this chapter, a new method for the automatic tuning of decentralised PI controllers for multivariable processes, based on GAs, was proposed. The major advantage of the proposed tuning method is the ability to handle arbitrary performance specifications in the time-domain, that can be different for each system output and set point test pattern combination. This is achieved by transforming the PI tuning problem into a function optimisation problem, using objective function J_M developed in Chapter 3. The method of adaptive fitness sharing developed in Chapter 4 is employed, in order to maximise the diversity of the obtained family of strictly optimal PI controllers. The numerical robustness and open architecture of GAs make the method applicable to the automatic tuning of a wide range of linear or non-linear parametric multivariable controllers, and not just PI controllers. The effectiveness of the proposed tuning method was supported by simulation results using three two-input, two-output processes with different degrees of interaction between the two loops. It was shown that, in all cases, the resulting PI controllers were strictly optimal, and thus the closed-loop systems completely satisfied all performance specifications. The method of adaptive fitness sharing was shown to achieve a high degree of diversity in the obtained family of solutions. The choice of GAs as a suitable optimisation method was supported by comparing GAs with two conventional optimisation methods, where it was shown that GAs have significantly higher success rates and are more immune to noise. A disadvantage of the proposed method in its current form is that it may only be useful in practice for off-line parametric controller tuning, mainly because of the stochastic nature of GAs and the relatively large number of closed-loop tests involved. Hence, a suitable process model and the use of a simulator may be required.

6 Solution to the Shell Standard Control Problem Using Genetically Tuned PID Controllers

6.1 Introduction

In this chapter, a solution to the Shell standard control problem is presented, based on genetic algorithms. The proposed scheme includes two discrete-time PID controllers with integral anti-windup and a multivariable Smith predictor to provide the required process output regulation, while the process input minimisation problem is analytically solved on-line, by estimating the unmeasured disturbances entering the process and solving the associated linear program. This, as well as the presence of constraints in the process manipulated variables, results in a complex, non-linear closed-loop system and hence, the manual tuning of the PID controllers according to some given performance specifications becomes a difficult task. Genetic algorithms are successfully applied to the automatic tuning of the PID controllers according to the given specifications, using an extension of the objective function J_M developed in Chapter 3. Simulation results are presented to demonstrate the effectiveness of the proposed control scheme.

6.2 The Shell Standard Control Problem

The Shell standard control problem was first published by the company in 1986 in the 1st Shell Process Control Workshop (Prett and Morari, 1987), with the intention to provide a standard and realistic test bed for the evaluation of new control theories and technologies. It captures most of the relevant control issues while staying as realistic as possible. The full problem statement and the model of the process under control is given in Appendix A, and can also be found in Prett and Morari (1987) and Prett, García, and Morari (1990). The process is a multivariable heavy oil fractionator (5-input, 7-output)

which is highly constrained, with very strong interactions and large dead times. The key elements of the Shell standard control problem are shown in Fig. 6.1 below.

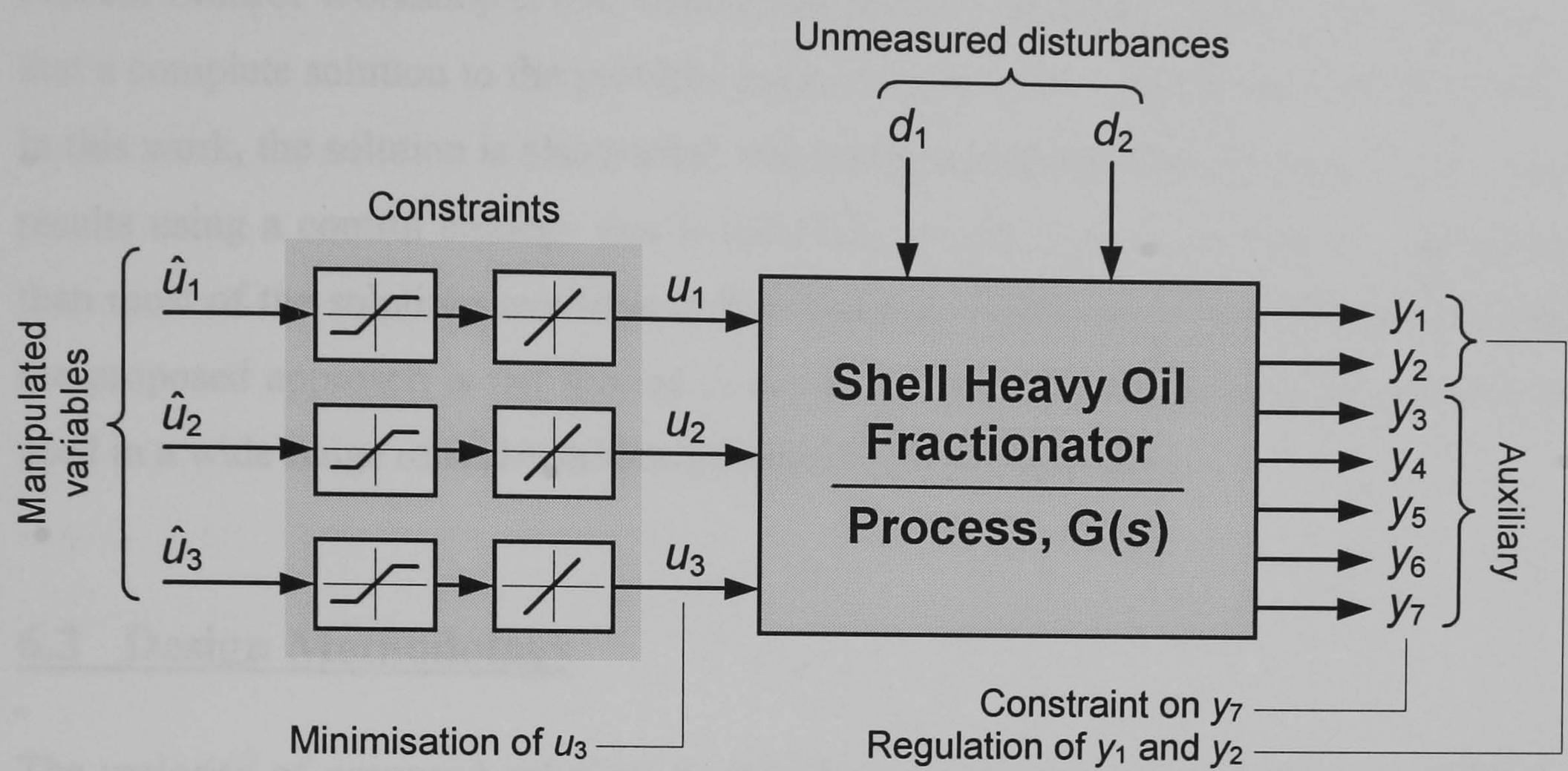


Fig. 6.1 *Key elements of the Shell standard control problem*

The problem is stated such that an infinite number of scenarios can occur in controlling the unit. The process input/output relations are linearly modelled using a matrix of first-order dead time transfer functions. Inputs u_1, u_2, and u_3 can be used as manipulated variables to control the process, but are subject to saturation (±0.5) and rate limit (±0.05 per minute) actuator hard constraints, thus making the process non-linear. Inputs d_1 and d_2 are unmeasured but bounded disturbances entering the process, with $|d_1| \leq 0.5$ and $|d_2| \leq 0.5$. Furthermore, the process is subject to uncertainties in the gains of the model transfer functions.

The main objective is to maintain process outputs y_1 and y_2 at specification (0.0 ±0.005 in the steady state), while at the same time input u_3 has to be minimised and output y_7 has to be kept to values of at least –0.5 at all times. Furthermore, output y_1 must be maintained within the maximum and minimum values of 0.5 and –0.5 at all times, and the unmeasured disturbances d_1 and d_2 have to be rejected even when the sensors of y_1 and y_2 fail. The closed-loop speed of response must be kept between 0.8 and 1.25 of the open-loop process bandwidth and the fastest permissible sampling time is 1 minute.

It is apparent that the Shell standard control problem is an extremely difficult problem which includes many possibly conflicting process requirements that are very difficult to satisfy. A number of partial solutions to the problem have been proposed in the 2nd Shell Process Control Workshop (Prett, García, and Morari, 1990) and it has been conjectured that a complete solution to the problem does not exist (Prett, García, and Morari, 1990). In this work, the solution is also partial, but has the advantage that it achieves very good results using a control strategy that is relatively simple and much easier to implement than most of the solutions available in the literature. Furthermore, it will be shown that the proposed approach is not limited to the Shell standard control problem and can be used in a wide range of real-world multivariable control problems.

6.3 Design Methodology

The majority of proposed solutions to the Shell standard control problem available in the literature utilise the state-of-the-art Quadratic Dynamic Matrix Control (QDMC) algorithm (García and Morshedi, 1986) developed by Shell. The main advantage of QDMC is that the objectives and constraints associated with the control problem are directly embedded in the process control algorithm, thus requiring only minimal *ad hoc* controller adjustments (García, Prett, and Morari, 1989). However, a disadvantage of QDMC is that it is extremely computationally intensive, and this is one of the reasons why it has not enjoyed widespread use in the small- and medium-size industries. In this work, PID controllers are employed as the main elements of the proposed solution. PID controllers are still widely used in industry, they are relatively easy to implement, and the majority of control personnel are familiar with their operation.

6.3.1 The Output Regulation Problem

Two discrete-time PID controllers with integral anti-windup loops and derivative term filtering were employed, to provide the integral actions necessary in order to achieve the regulation requirement for outputs y_1 and y_2. Of the three manipulated variables, u_1 and u_2 were chosen for closing the two PID loops, since u_3 has the additional minimisation requirement and hence cannot be used in the loops. Furthermore, the choice of u_1 and u_2 to control y_1 and y_2 arises naturally from process operation considerations. This results in the following 2×2 transfer function matrix (see Appendix A).

$$\mathbf{G}_R(s) = \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} = \begin{bmatrix} \dfrac{4.05e^{-27s}}{50s+1} & \dfrac{1.77e^{-28s}}{60s+1} \\ \dfrac{5.39e^{-18s}}{50s+1} & \dfrac{5.72e^{-14s}}{60s+1} \end{bmatrix} \tag{6.1}$$

By examining the elements of $\mathbf{G}_R(s)$ it is observed that the best pairing of manipulated and controlled variables is to control y_1 with u_1 and y_2 with u_2, since the gains in the main diagonal of $\mathbf{G}_R(s)$ are sufficiently large and thus the interaction between the loops will be minimal. Note, however, that the gain of $G_{21}(s)$ is larger than that of $G_{11}(s)$ and hence, there will be strong interaction from loop 1 to loop 2. The above input/output pairing is also indicated by the steady-state relative gain array of $\mathbf{G}_R(s)$, which consists of positive elements in its main diagonal and negative elements elsewhere.

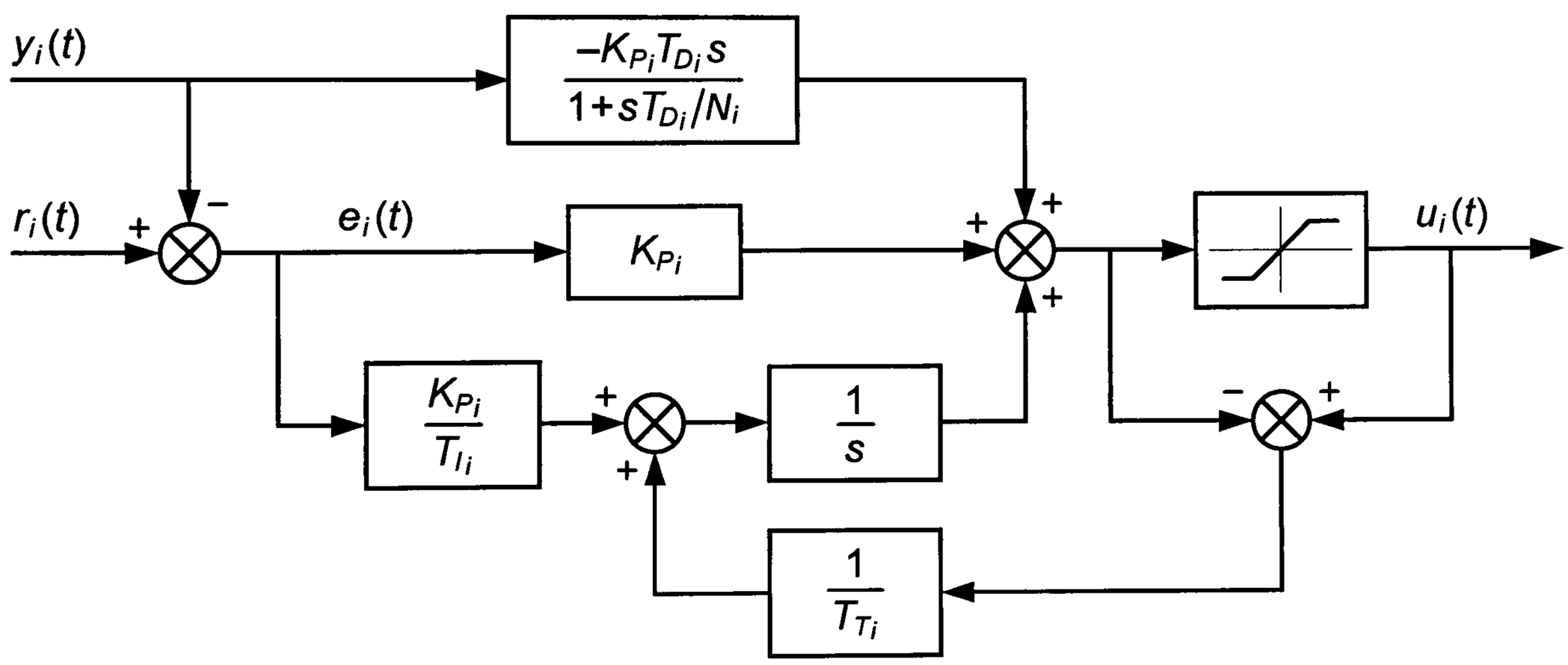


Fig. 6.2 PID controller in loop i with integral anti-windup (continuous-time)

The PID controller structure used in this work is exactly the one proposed by Åström and Hägglund (1995), and is shown in Fig. 6.2. The PID controllers are discrete-time, arranged in a decentralised (diagonal) structure. When there is no actuator saturation, the output $u_i(z)$ of controller i is given by

$$u_i(z) = \overbrace{K_{P_i}e_i(z)}^{\text{Proportional}} + \overbrace{\frac{TK_{P_i}(z+1)}{2T_{I_i}(z-1)}e_i(z)}^{\text{Integral}} + \overbrace{\frac{-K_{P_i}T_{D_i}N_i(z-1)}{T_{D_i}(z-1)+N_iTz}y_i(z)}^{\text{Derivative}} \tag{6.2}$$

where $e_i(z) = r_i(z) - y_i(z)$ is the error signal for loop i, $r_i(z)$ is the set point signal for loop i, and $i = 1, 2$. The discrete-time transfer functions for the integral and derivative terms shown in Eq. (6.2) were obtained using Tustin's approximation and the backward difference approximation, respectively. With reference to Fig. 6.2 and Eq. (6.2), K_{P_i}, T_{I_i}, and T_{D_i} denote the parameters of the i-th PID controller, and T denotes the discretization sample time. Parameter N_i is used to limit the high-frequency gain of the derivative term, thus improving the high-frequency noise immunity of the controller. This is achieved by filtering the derivative signal using a first-order low-pass filter with time constant T_{D_i}/N_i. Typical values of N_i range from 8 to 20 (Åström and Hägglund, 1995). In this work, the setting $N_1 = N_2 = 8$ was used, resulting in a filter with the lowest cut-off frequency in this range. Parameter T_{T_i} in Fig. 6.2 is known as the tracking time constant and controls the effect of the integral anti-windup mechanism. Parameter T_{T_i} should be larger than T_{D_i} and smaller than T_{I_i}. A rule of thumb is to choose $T_{T_i} = \sqrt{T_{I_i} T_{D_i}}$ (Åström and Hägglund, 1995). This, however, has the serious disadvantage that it can result in arbitrarily large signals in the anti-windup feedback loop when T_{D_i} tends to zero, which can destabilise the loop. Specifically, in digital implementations of the PID controller, such as the one shown in Eq. (6.2), the anti-windup loop in Fig. 6.2 becomes unstable when $T_{T_i} \leq T/2$. To see this, consider the anti-windup loop shown in Fig. 6.3 below.

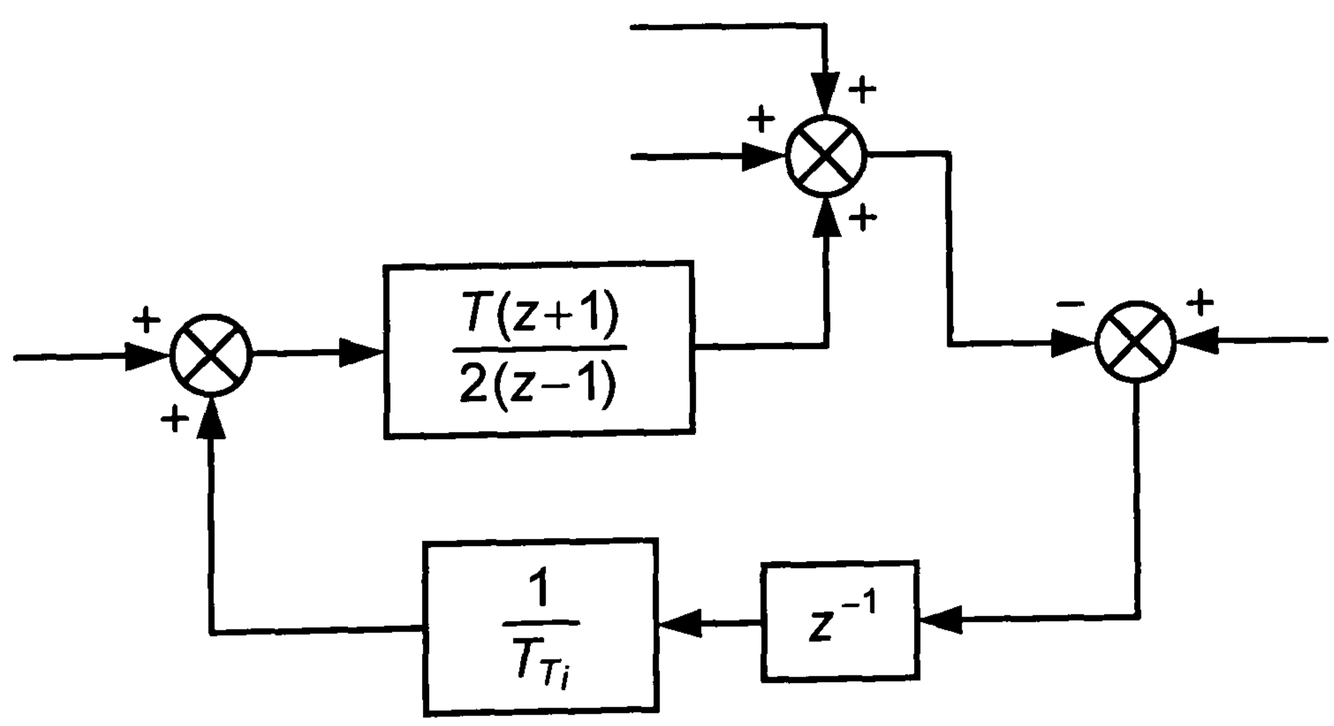


Fig. 6.3 Digital implementation of the anti-windup loop

The discrete-time integrator in Fig. 6.3 is obtained using Tustin's approximation. The unit delay in the feedback path is necessary, in order to eliminate the algebraic loop that

would otherwise be formed, since the current output of the integrator is a function of the current input. When the actuator saturates, the main PID loop is broken, and the external signals entering the anti-windup loop are not affected by the signals generated inside the loop. Therefore, the stability of the anti-windup loop solely depends on its loop transfer function, whose characteristic equation is shown below.

$$2T_{T_i}z^2+\left(T-2T_{T_i}\right)z+T=0 \tag{6.3}$$

In order for the above polynomial equation to have all its roots inside the unit circle, the following three conditions must be satisfied (Jury, 1973).

$$\begin{aligned} T &> 0 \\ T_{T_i} &> 0 \\ 2T_{T_i} &> |T| \end{aligned} \tag{6.4}$$

Clearly, all three conditions are satisfied for positive T, only when $T_{T_i} > T/2$. It is easy to see that this result is also true for the forward and backward difference approximations, and shows that the setting $T_{T_i} = \sqrt{T_{I_i}T_{D_i}}$ recommended by Åström and Hägglund (1995) must be used with caution when T_{D_i} is small. The above result has also been confirmed by personal communication with Hägglund (1999).

In order to avoid this instability problem, the setting $T_{T_i} = \max\left\{\sqrt{T_{I_i}T_{D_i}}, T\right\}$ was used in this work, which ensures that $T_{T_i} \geq T$ even when the derivative term is switched off. This lower bound on T_{T_i} ensures that the anti-windup loop is always stable and well-damped, irrespective of the values of K_{P_i}, T_{I_i}, and T_{D_i}.

6.3.2 The Input Minimisation Problem

The minimisation of process input u_3 is a challenging problem since the optimal value of u_3 does not remain fixed, but changes as the unmeasured disturbances d_1 and d_2 change. Hence, the problem cannot be solved by conventional control designs such as a PID controller. The optimal value of u_3 is defined as the lowest possible value of u_3 such that the closed-loop system satisfies all control objectives without violating any of

the control constraints. This requirement should be satisfied during the entire process operating time. Formally, this results in the following optimisation problem.

$$\min_{u_i(t)} u_3(t), \ \forall t, \ i=1,2,3$$

$$\begin{aligned} \text{Subject to:} \quad |u_i(t)| &\leq 0.5, \quad \forall t, \ i=1,2,3 \\ \left|\frac{du_i(t)}{dt}\right| &\leq 0.05, \quad \forall t, \ i=1,2,3 \\ |y_i(t)| &\leq 0.005, \quad t \geq t_s, \ i=1,2 \\ |y_1(t)| &\leq 0.5, \quad \forall t \\ y_7(t) &\geq -0.5, \quad \forall t \end{aligned} \tag{6.5}$$

where t_s is the time required for the closed-loop system to reach the steady state, and must be chosen such that the closed-loop system speed of response is between 0.8 and 1.25 of the open-loop process bandwidth. This problem is difficult to solve analytically since all signals involved in (6.5) are not constant, but vary with time. The requirements for u_i can be satisfied *a priori* by simply constraining the controller outputs to the level and rate limits of ±0.5 and ±0.05 per minute, respectively. This is permissible because of the anti-windup mechanism employed in the PID controllers, which ensures that no integral wind-up will occur. Assuming that output constraint violations are allowed in the transient period, the optimisation problem defined in (6.5) can be solved analytically for the *steady-state* optimal value of u_3 as follows.

Let K_{ij} denote the steady-state gain of the i, j element of the process transfer function matrix $\mathbf{G}(s)$. Then, in the steady state, the following equation holds.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_7 \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} & K_{15} \\ K_{21} & K_{22} & K_{23} & K_{24} & K_{25} \\ K_{71} & K_{72} & K_{73} & K_{74} & K_{75} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ d_1 \\ d_2 \end{bmatrix} \tag{6.6}$$

In the steady state, outputs y_1 and y_2 are guaranteed to be zero because of the integral action of the two PID controllers. Furthermore, it is easy to see that, in the steady state, minimisation of u_3 means that *at least one* of u_1, u_2, u_3, y_7 will be exactly at its

constraint boundary. If the disturbances d_1 and d_2 are known, Eq. (6.6) is simply a system of three simultaneous linear equations with four unknowns (u_1, u_2, u_3, y_7). It is, therefore, sufficient to set each one of u_1, u_2, u_3, y_7 to its constraint boundaries (one at a time) and use Eq. (6.6) to solve for the remaining three. Specifically, $u_1 \in \{+0.5, -0.5\}$, $u_2 \in \{+0.5, -0.5\}$, $u_3 \in \{+0.5, -0.5\}$, and $y_7 \in \{-0.5\}$. Hence, Eq. (6.6) has to be solved seven times and this will result in seven different values for u_3. The optimal value of u_3 is the minimum of these values for which *none* of the four variables, u_1, u_2, u_3, y_7, violates the constraints. This is a standard linear programming (LP) problem and it can be shown that it always has a feasible solution when $|d_1| \leq 0.5$ and $|d_2| \leq 0.5$ (see Section 6.3.3). It should be pointed out at this point that if the gain matrix in Eq. (6.6) is not constant, as in the case of uncertainties in the gains of the model, the resulting value for u_3 may not be optimal. In this case, one could specify more conservative constraint boundaries, so that the effects of uncertainties are accounted for (see Section 6.5.2).

In the solution of the minimisation problem (6.5) described above, it was assumed that output constraint violations are allowed in the transient period. It will be shown in Section 6.4.2 that the transient requirements for process outputs y_1 and y_2 can be used as closed-loop performance specifications for the tuning of the two PID controllers, by means of an extension of objective function J_M.

6.3.3 LP Solution Feasibility Analysis

It was shown in the previous section that, in the steady state, optimisation problem (6.5) is equivalent to the following linear program.

$$\min_{u_i} u_3, \quad i = 1, 2, 3$$

$$\text{Subject to:} \quad \begin{aligned} |u_i| &\leq 0.5, \quad i = 1, 2, 3 \\ |y_i| &= 0, \quad i = 1, 2 \\ y_7 &\geq -0.5 \end{aligned} \tag{6.7}$$

It is known from linear programming theory (Wood and Dantzig, 1949; Dantzig, 1949) that if a feasible solution of a linear program exists, it must lie precisely at the edges of the convex polyhedron defined by the inequalities associated with the linear program.

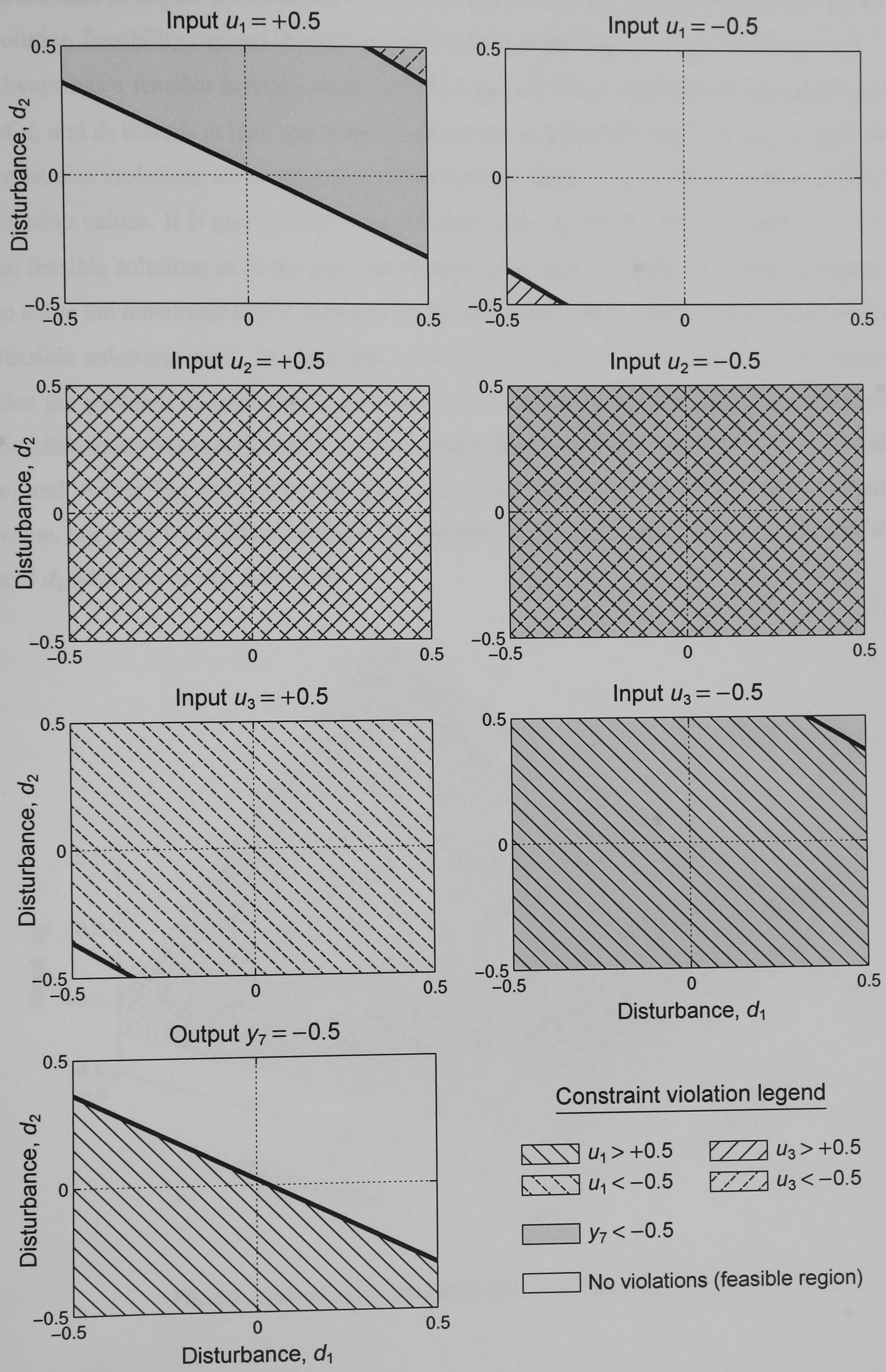


Fig. 6.4 *Feasibility regions of linear programming problem (6.7)*

In the case of the LP problem shown in (6.7), there are seven linear inequalities, and the solution feasibility regions in each case are shown graphically in Fig. 6.4. Clearly, (6.7) always has a feasible solution when $|d_1| \leq 0.5$ and $|d_2| \leq 0.5$, because for any given pair of d_1 and d_2 there is at least one case which results in a feasible value for u_3 (i.e. with no constraint violations on u_1, u_2, u_3, y_7). The optimal value of u_3 is the minimum of these feasible values. It is also clearly observed that when $u_2 = +0.5$, $u_2 = -0.5$, and $u_3 = -0.5$, no feasible solutions exist for any combination of d_1 and d_2. Hence, it is only necessary to solve the remaining four systems of linear equations which correspond to cases where feasible solutions exist. The fact that no feasible solutions exist when $u_3 = -0.5$ means that the problem of minimising u_3 cannot be solved by simply setting u_3 at its lowest possible value because this will result in steady-state constraint violations on y_7, or both u_1 and y_7 (see Fig. 6.4). It is, therefore, concluded that u_3 cannot be kept at a constant value, but must be adjusted on-line. The solution of LP problem (6.7) as a function of d_1 and d_2 is shown in Fig. 6.5 below.

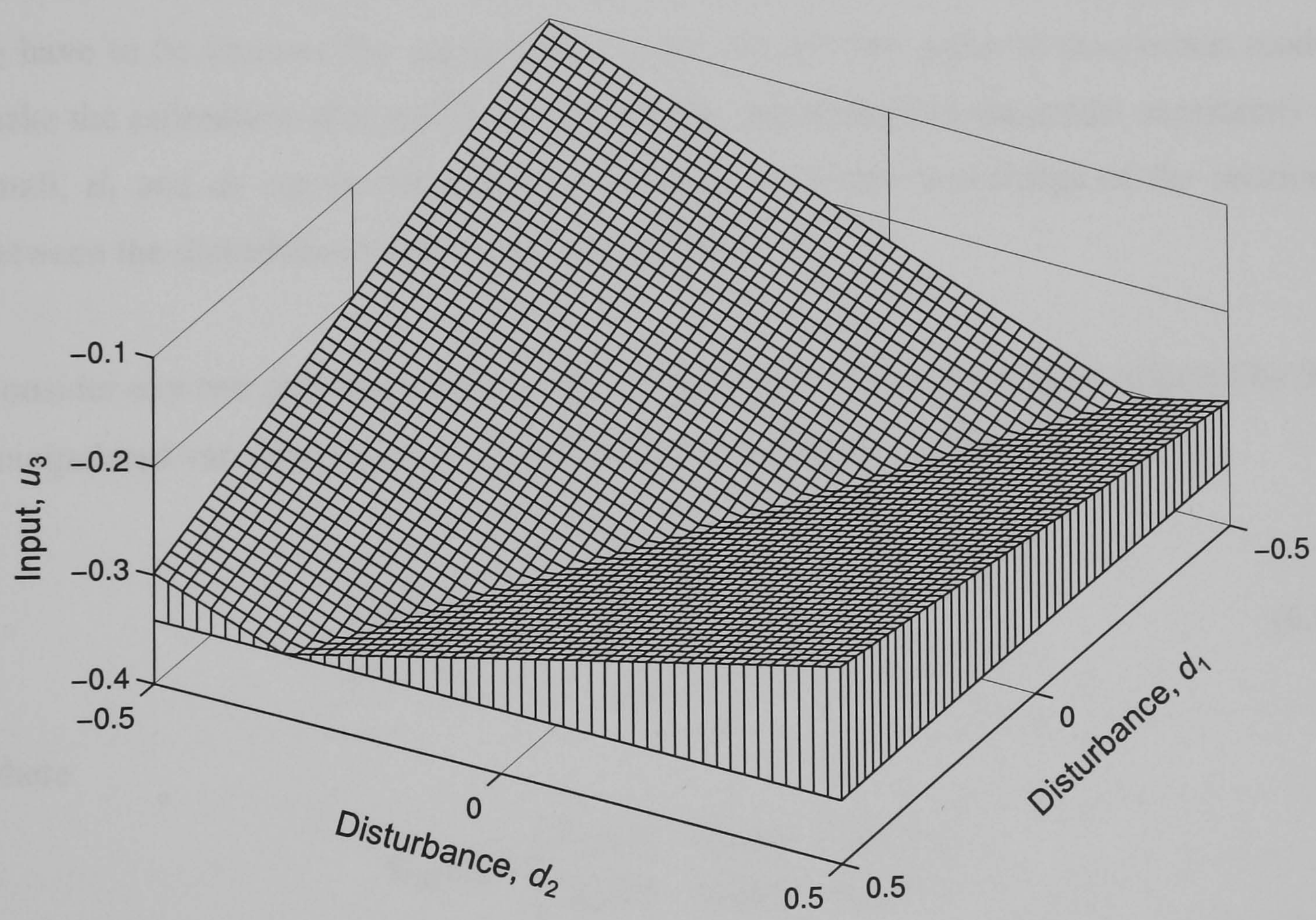


Fig. 6.5 *Optimal steady-state value of u_3 as a function of d_1 and d_2*

The minimum achievable steady-state value of u_3 is $u_3 = -0.3419$, and is achieved when $d_1 = +0.5$ and $d_2 = -0.3152$. The worst-case scenario is when $d_1 = d_2 = -0.5$, where the

minimum achievable steady-state value of u_3 is $u_3 = -0.1040$. The optimal steady-state values of u_3 for various combinations of d_1 and d_2 are shown in Table 6.1 below.

TABLE 6.1 Optimal steady-state values of u_3 for various disturbance vectors $\mathbf{d} = [d_1\ d_2]^T$

d_1	d_2	Optimal value of u_3	Constraint boundary condition
+0.5	+0.5	–0.2178	Output y_7 at boundary ($y_7 = -0.5$)
–0.5	–0.5	–0.1040	Input u_1 at upper boundary ($u_1 = +0.5$)
+0.5	–0.5	–0.2988	Input u_1 at upper boundary ($u_1 = +0.5$)
–0.5	+0.5	–0.2834	Output y_7 at boundary ($y_7 = -0.5$)
0	0	–0.3181	Input u_1 at upper boundary ($u_1 = +0.5$)
+0.5	–0.3152	–0.3419	Output y_7 at boundary ($y_7 = -0.5$)

6.3.4 Unmeasured Disturbance Estimation

In order to be able to solve the LP problem in (6.7), the unmeasured disturbances d_1 and d_2 have to be known. The uncertainties in the steady-state gains of the process model make the estimation of d_1 and d_2 a difficult task. Assuming that the model uncertainty is small, d_1 and d_2 can be estimated by using the available knowledge of the relations between the disturbances and the process outputs.

Consider any two process outputs y_a and y_b, with $a < b$. These outputs are affected by the manipulated variables (known) as well as the disturbances (unknown).

$$\begin{bmatrix} y_a(s) \\ y_b(s) \end{bmatrix} = \mathbf{G}_M(s) \begin{bmatrix} u_1(s) \\ u_2(s) \\ u_3(s) \end{bmatrix} + \mathbf{G}_D(s) \begin{bmatrix} d_1(s) \\ d_2(s) \end{bmatrix} \tag{6.8}$$

where

$$\mathbf{G}_M(s) = \begin{bmatrix} G_{a1}(s) & G_{a2}(s) & G_{a3}(s) \\ G_{b1}(s) & G_{b2}(s) & G_{b3}(s) \end{bmatrix}$$

$$\mathbf{G}_D(s) = \begin{bmatrix} G_{a4}(s) & G_{a5}(s) \\ G_{b4}(s) & G_{b5}(s) \end{bmatrix} \tag{6.9}$$

Transfer function matrices $\mathbf{G}_M(s)$ and $\mathbf{G}_D(s)$ are subsystems of process model $\mathbf{G}(s)$ given in Appendix A, and they map manipulated variables u_1, u_2, u_3 and disturbances d_1

and d_2, respectively, to process outputs y_a and y_b. Therefore, the contribution of u_1, u_2, and u_3 (but not d_1 and d_2) to y_a and y_b can be expressed as

$$\begin{bmatrix} \hat{y}_a(s) \\ \hat{y}_b(s) \end{bmatrix} = \mathbf{G}_M(s) \begin{bmatrix} u_1(s) \\ u_2(s) \\ u_3(s) \end{bmatrix} \tag{6.10}$$

An estimate of the disturbance vector can be computed by subtracting Eq. (6.10) from Eq. (6.8) and solving for $[d_1\ d_2]^T$, as shown below.

$$\begin{bmatrix} d_1(s) \\ d_2(s) \end{bmatrix} = \mathbf{G}_D^{-1}(s) \begin{bmatrix} y_a(s) - \hat{y}_a(s) \\ y_b(s) - \hat{y}_b(s) \end{bmatrix} \tag{6.11}$$

The direct inversion of $\mathbf{G}_D(s)$ results in an unrealisable system, and this is true for all possible choices of a and b. This situation can be resolved by considering $\mathbf{G}_D(z)$, the z-Transform of $\mathbf{G}_D(s)$. Transfer function matrix $\mathbf{G}_D(z)$ must admit the following representation.

$$\mathbf{G}_D(z) = \begin{bmatrix} G_{a4}^*(z) & G_{a5}^*(z) \\ G_{b4}^*(z) & G_{b5}^*(z) \end{bmatrix} z^{-k} \tag{6.12}$$

where $G_{ij}^*(z)$ denotes the part of $G_{ij}(z)$ that remains when all delay states (including the one associated with the zero-order hold) have been removed. The inverse of $\mathbf{G}_D(z)$ can now be expressed as

$$\mathbf{G}_D^{-1}(z) = \begin{bmatrix} G_{a4}^*(z) & G_{a5}^*(z) \\ G_{b4}^*(z) & G_{b5}^*(z) \end{bmatrix}^{-1} z^{k} \tag{6.13}$$

Multiplying the above equation by z^{-k} (i.e. removing the common factor z^k) results in a realisable system which can be used to estimate disturbance vector $\mathbf{d} = [d_1\ d_2]^T$ at each sampling instant. Note that representation (6.12) is only possible if all elements of $\mathbf{G}_D(z)$ have the same number of delay states. It can be seen by examining the delays of the Shell process model that this is true when $a, b \in \{3, \ldots, 7\}$. This permits $\binom{5}{2} = 10$ different combinations for a and b. The number of common delay states of $\mathbf{G}_D(z)$ in all combinations is $k=1$, which means that there will be a delay of one sample in the disturbance estimates. Pairs ($a=3$, $b=7$) and ($a=4$, $b=6$) are rejected because they

result in $\mathbf{G}_D(s)$ being non-minimum phase and therefore inverse unstable. Among the remaining eight combinations of a and b, pair ($a=3$, $b=5$) was chosen because the estimator obtained exhibits the lowest sensitivity to measurement noise. If noise exists in the measurements of y_a and y_b, Eq. (6.11) can be written as

$$\begin{aligned}\begin{bmatrix} d_1(s) \\ d_2(s) \end{bmatrix} &= \mathbf{G}_D^{-1}(s) \begin{bmatrix} y_a(s)+e_a(s)-\hat{y}_a(s) \\ y_b(s)+e_b(s)-\hat{y}_b(s) \end{bmatrix} \\ &= \mathbf{G}_D^{-1}(s)\mathbf{e}(s) + \mathbf{G}_D^{-1}(s) \begin{bmatrix} y_a(s)-\hat{y}_a(s) \\ y_b(s)-\hat{y}_b(s) \end{bmatrix}\end{aligned} \tag{6.14}$$

where $\mathbf{e}=[e_a\ e_b]^T$ is the measurement noise vector. It is observed from Eq. (6.14) that the contribution of the noise to the disturbance estimates is $\mathbf{G}_D^{-1}(s)\mathbf{e}(s)$. Therefore, the sensitivity to measurement noise can be minimised by selecting a and b such that the resulting $\mathbf{G}_D^{-1}(s)$ exhibits the lowest 'gain'. Although a transfer function matrix does not have a unique gain, the largest singular value of $\mathbf{G}_D^{-1}(s)$, denoted $\bar{\sigma} \equiv \|\mathbf{G}_D^{-1}\|$, can be used as it provides an upper bound on the amplification of $\|\mathbf{e}\|$ by $\mathbf{G}_D^{-1}$. The largest singular values of $\mathbf{G}_D^{-1}(s)$, with $s=j\omega$, for all eight combinations of a and b are shown in Fig. 6.6. It can be seen that the chosen pair (indicated by the solid thick line) exhibits the lowest amplification of noise almost over the entire frequency range of interest. Furthermore, since $a, b \notin \{1, 2\}$, the measurements of y_1 and y_2 are not required for the operation of the estimator. Hence, the disturbance estimates will not be affected in case of failure in one or both of the sensors of y_1 and y_2. The estimator in its final form is given below.

$$\begin{bmatrix} \hat{d}_1(z) \\ \hat{d}_2(z) \end{bmatrix} = z^{-k}\mathbf{G}_F(z)\,\mathbf{G}_D^{-1}(z) \left\{ \begin{bmatrix} y_a(z) \\ y_b(z) \end{bmatrix} - \mathbf{G}_M(z) \begin{bmatrix} u_1(z) \\ u_2(z) \\ u_3(z) \end{bmatrix} \right\} \tag{6.15}$$

where $a=3$, $b=5$, and $\mathbf{G}_F(z)$ is a diagonal matrix of two discrete-time low-pass filters, which can be used to reject noise. In this work, $\mathbf{G}_F(z)=\mathbf{I}$ was used (i.e. no filtering). The obtained estimates of d_1 and d_2 at each sampling instant are then used to compute the optimal steady-state value of u_3 on-line, as described in Section 6.3.2. An additional low-pass filter may be employed to condition u_3 before it is applied to the process. In this work, a first-order filter was used with a time constant of 50 minutes.

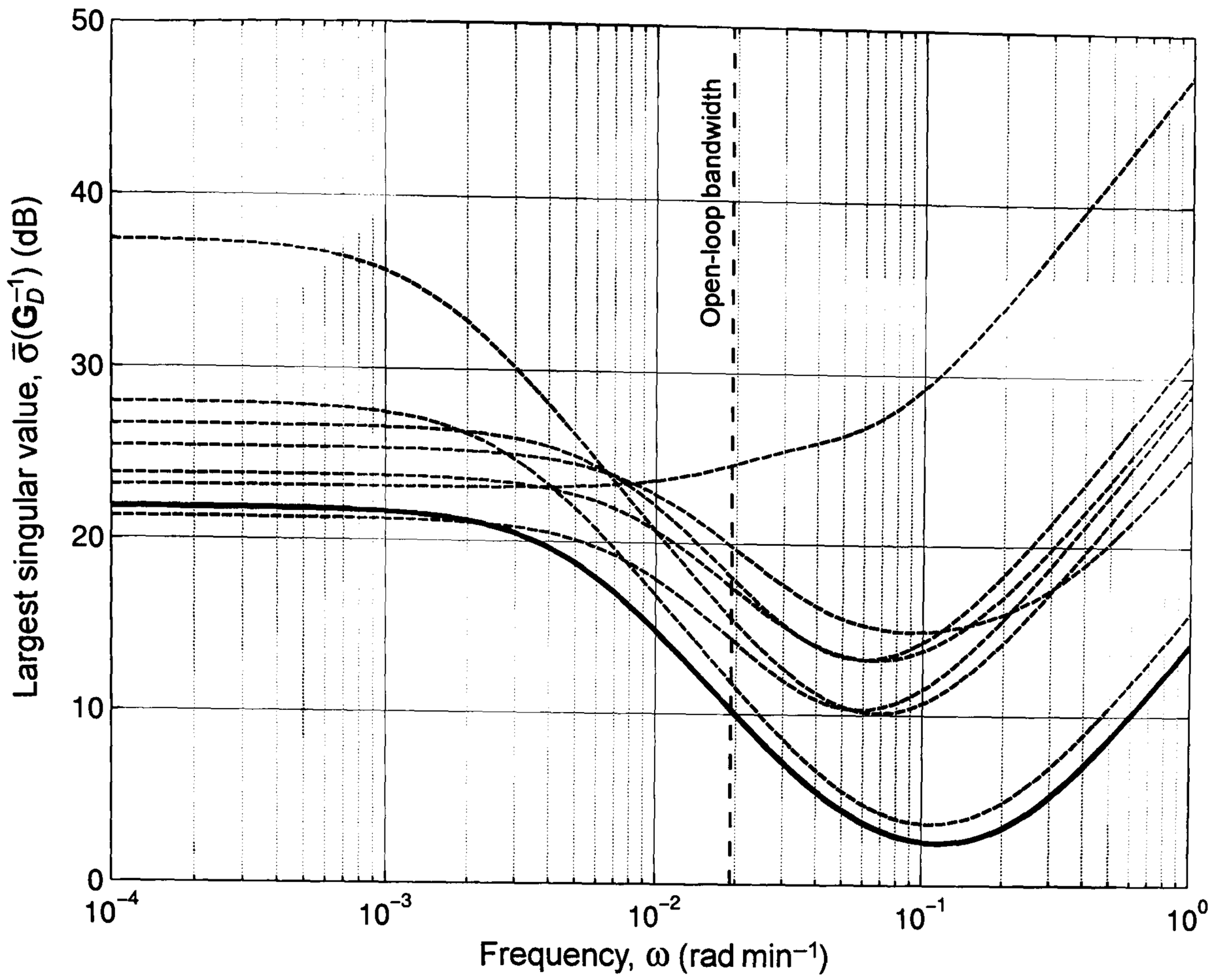


Fig. 6.6 *Largest singular value frequency responses of $\mathbf{G}_D^{-1}(s)$ for the eight permissible combinations of a and b (solid thick line indicates the chosen pair a=3, b=5)*

Note that care should be taken when using disturbance estimators such as the one given in Eq. (6.15) as part of an on-line control algorithm, because they can be very sensitive to modelling errors and can thus produce unreliable disturbance estimates when large uncertainties in the model parameters are present. Other, more accurate disturbance estimation methods (such as a Kalman filter, for example) may be required, depending on the complexity of the controlled process and the accuracy of the available model.

6.3.5 Multivariable Smith Predictor Control

The large dead times associated with $\mathbf{G}_R(s)$ limit the achievable performance of the PID controllers discussed in Section 6.3.1. To compensate for this, a multivariable Smith predictor proposed by Maciejowski (1994) was used in this work. This is an extension of the SISO approach proposed by Smith (1957). The multivariable Smith predictor structure in the form of Internal Model Control (IMC) is shown in Fig. 6.7.

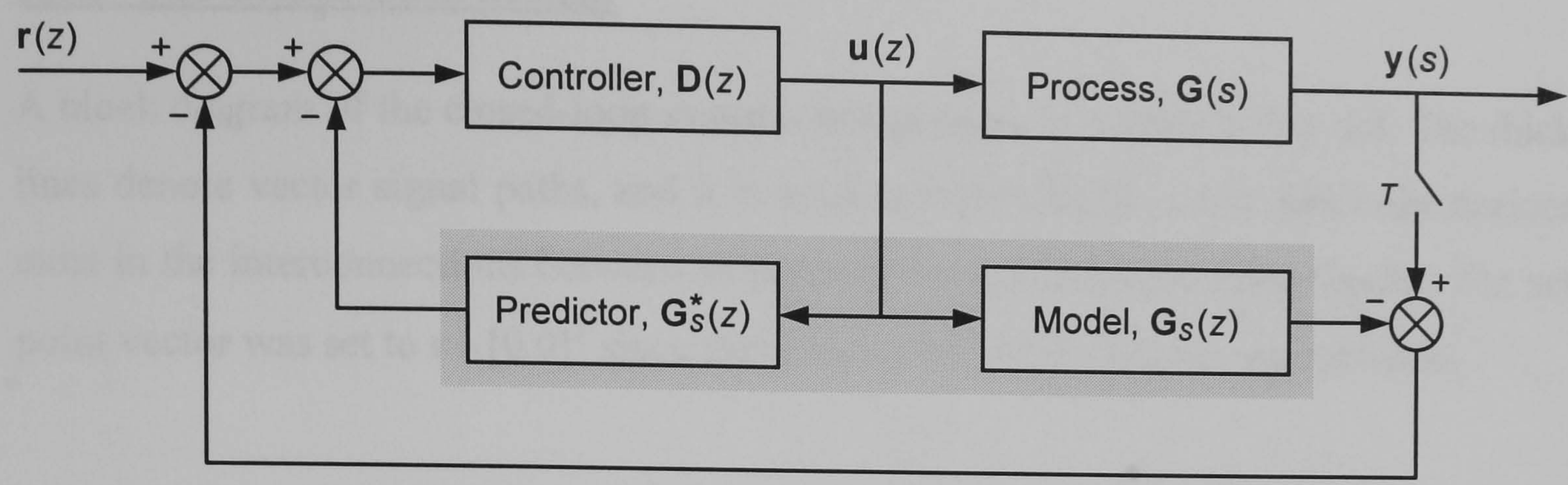


Fig. 6.7 Multivariable Smith predictor in the form of IMC (Maciejowski, 1994)

The Smith predictor consists of a discrete-time process model $\mathbf{G}_S(z)$, and a delay-free model $\mathbf{G}_S^*(z)$ obtained by removing all delay states from all of the elements of $\mathbf{G}_S(z)$. It is observed that in the Smith predictor structure shown in Fig. 6.7, the controller $\mathbf{D}(z)$ is controlling the delay-free model and not the actual process. Hence, the effect of the controller output vector $\mathbf{u}$ to the process output vector $\mathbf{y}$ is not delayed by the inherent process delays, but is immediately available to the controller. In this way, processes with large dead times such as $\mathbf{G}_R(s)$ can be controlled by conventional controllers such as PID controllers without significant loss in achievable performance. An auxiliary loop (outer loop) compensates for modelling errors by appropriately modifying the set point vector $\mathbf{r}$. The two process models associated with the Smith predictor can easily be obtained by discretizing the transfer function matrix $\mathbf{G}(s)$ of the Shell process. Since there are three manipulated variables, u_1, u_2, and u_3, and two controlled variables, y_1 and y_2, the process model $\mathbf{G}_S(z)$ and the predictor $\mathbf{G}_S^*(z)$ are represented by the 2×3 transfer function matrices shown below.

$$\mathbf{G}_S(z) = \begin{bmatrix} G_{11}(z) & G_{12}(z) & G_{13}(z) \\ G_{21}(z) & G_{22}(z) & G_{23}(z) \end{bmatrix}$$
$$\mathbf{G}_S^*(z) = \begin{bmatrix} G_{11}^*(z) & G_{12}^*(z) & G_{13}^*(z) \\ G_{21}^*(z) & G_{22}^*(z) & G_{23}^*(z) \end{bmatrix} \tag{6.16}$$

where $G_{ij}^*(z)$ denotes the part of $G_{ij}(z)$ that remains when all its delay states have been removed. Note that in this case there is no requirement for the number of delay states in the elements of $\mathbf{G}_S(z)$.

6.3.6 The Closed-Loop System

A block diagram of the closed-loop system configuration is shown in Fig. 6.8. The thick lines denote vector signal paths, and it is assumed that samplers and data hold devices exist in the interconnections between continuous-time and discrete-time blocks. The set point vector was set to $\mathbf{r} = [0\ 0]^T$ since there are no set point tracking requirements.

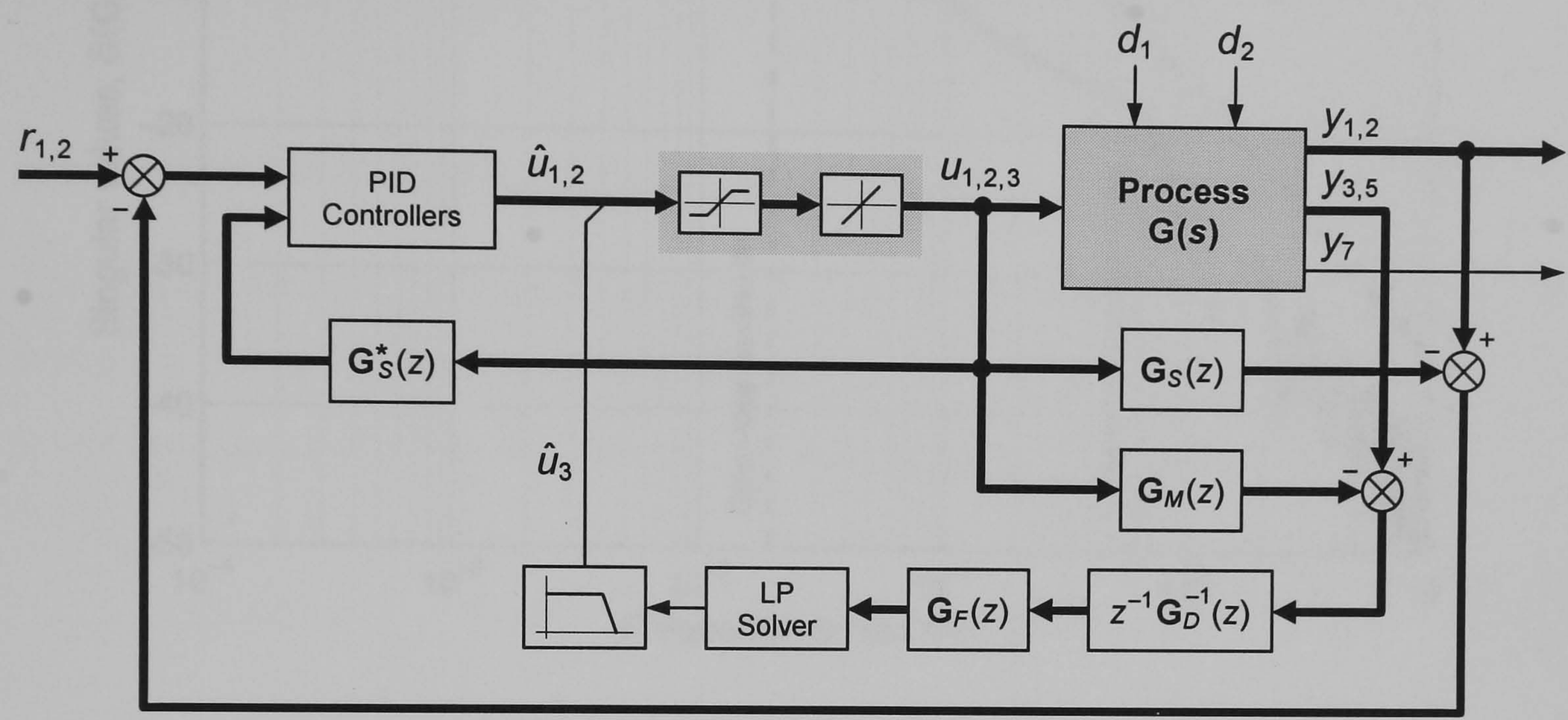


Fig. 6.8 *The closed-loop system configuration*

The sample time for all discrete-time blocks was chosen to be $T = 5$ minutes, which is five times the minimum permissible. The sample time imposes an upper bound on the achievable closed-loop bandwidth, which cannot exceed π/T radians per time unit, or 0.6283 rad min^{-1} in this case. Fig. 6.9 shows the singular value frequency responses of $\mathbf{G}_R(s)$, where it is observed that the closed-loop bandwidth upper bound imposed by T is well above the open-loop process bandwidth which is about 0.0192 rad min^{-1}.

Process manipulated variables u_1, u_2 and u_3 were constrained in accordance with the problem specifications. Specifically, $|u_i| \leq 0.5$ and $|\Delta u_i| \leq 0.05T$, where Δu_i denotes the change in u_i between two successive sampling instants, and $i = 1, 2, 3$. These constraints were regarded as equipment limitations, such that if the PID controllers demand values outside the given ranges, the limiting values are applied. The modified z-Transform was employed to obtain $\mathbf{G}_M(z)$, $\mathbf{G}_D^{-1}(z)$, $\mathbf{G}_S(z)$, and $\mathbf{G}_S^*(z)$, from the nominal process model $\mathbf{G}(s)$. The minimal state-space realisations of these systems are given in Appendix B.

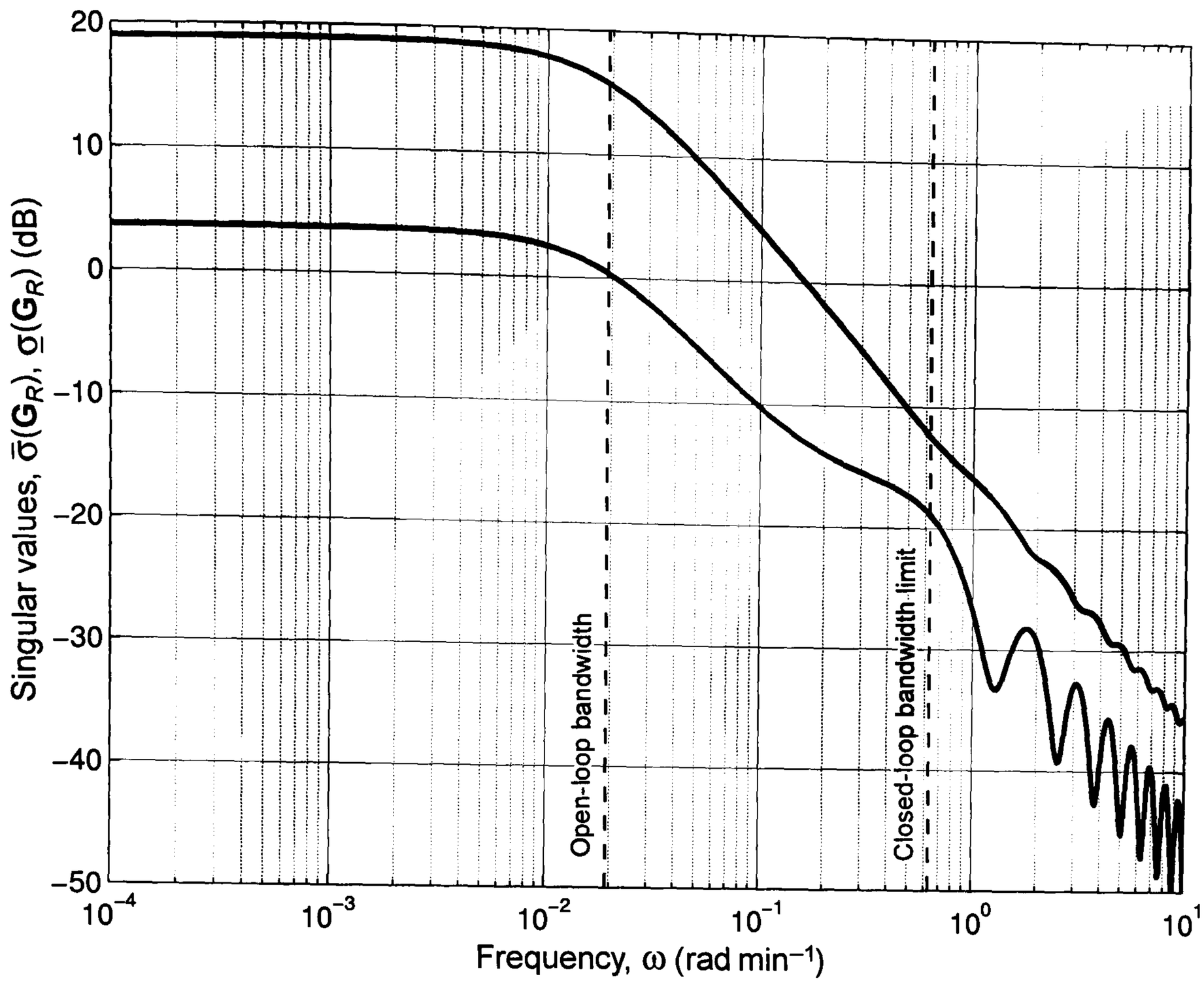


Fig. 6.9 *Singular values (frequency response) of* $\mathbf{G}_R(s)$

The saturation and rate limit non-linearities associated with the process, as well as the linear program solver used for the minimisation of u_3 result in a complex closed-loop system, and hence the tuning of the two PID controllers becomes a difficult task. In this work, the PID tuning problem is transformed into a function optimisation problem using an extension of the objective function J_M developed in Chapter 3, and then GAs are employed to search for suitable PID controllers to meet the design specifications for outputs y_1 and y_2. This is discussed in Section 6.4.

6.4 PID Controller Tuning

This section focuses on the tuning of the two PID controllers according to the design specifications. Objective function J_M is used as the basis for the controller performance index assignment strategy. In contrast to the tuning problems discussed in Chapter 5, in the case of the Shell standard control problem there are no set point tracking or loop coupling specifications. The output regulation objective is to reject disturbances d_1 and

d_2 and maintain outputs y_1 and y_2 at specification (0.0 ±0.005 in the steady state). There is also a transient requirement for y_1 which must be maintained within the maximum and minimum values of 0.5 and –0.5 at all t. Since the main requirement is to reject d_1 and d_2, the PID controller performance can be evaluated by subjecting the closed-loop system to a set of disturbance test patterns, and objective function J_M can still be used to compare the resulting responses for y_1 and y_2 with the problem specifications. This is discussed in detail in Section 6.4.1.

6.4.1 Objective Function Formulation

The performance of PID controller i under disturbance test pattern j can be quantified using the following function, where $\mathbf{D} \in \mathcal{D}$ and $f_{ij}^{(u)}(t)$ and $f_{ij}^{(l)}(t)$ are the upper and lower boundary functions representing the corresponding specifications.

$$O_{ij}(\mathbf{D}) := \int_0^{t_{max}} \left(\max\left\{ f_{ij}^{(l)}(t) - y_i(t), 0 \right\} + \max\left\{ y_i(t) - f_{ij}^{(u)}(t), 0 \right\} \right) dt \qquad (6.17)$$

This function is the same as J_{ij} in Eq. (3.16), and quantifies the controller performance, based solely on the shape of the process output signals. However, there may be cases where it is desirable or necessary to suppress the activity of the manipulated variables. A measure of the activity of manipulated variable i under disturbance test pattern j can be obtained by evaluating the following function.

$$I_{ij}(\mathbf{D}) := \sum_{k=1}^{k_{max}} \left(\frac{\Delta u_i(kT)}{\delta_i} \right)^2 \qquad (6.18)$$

where $\Delta u_i(kT) = u_i(kT) - u_i[(k-1)T]$, T denotes the sample time, and $k_{max} = t_{max}/T$. The manipulated variable moves Δu_i are normalised by dividing them with their absolute rate limits of δ_i units per sample. Similarly to the formulation of J_M, the terms O_{ij} and I_{ij} can be weighted and lumped together to form an objective function that can be used to quantify the performance of candidate controller $\mathbf{D}$ with respect to all combinations of i and j. Formally, the objective function is defined as

$$J_0(\mathbf{D}) := \max_j \left\{ \sum_i \left[\alpha_{ij} O_{ij}(\mathbf{D}) + \lambda_{ij} I_{ij}(\mathbf{D}) \right] \right\} \in \mathbb{R}_+ \qquad (6.19)$$

where $\alpha_{ij} \geq 0$ and $\lambda_{ij} \geq 0$ are the output and input weighting coefficients, respectively, for loop i under disturbance test pattern j. It is observed that the numerical evaluation of J_0 for one candidate controller requires the closed-loop system to be simulated as many times as there are disturbance patterns. In simulation run j, disturbance test pattern j is applied to the closed-loop system, and output and input terms O_{ij} and I_{ij}, respectively, are evaluated for all loops. The results are then weighted and added together to form a single number that indicates the quality of the candidate controller under disturbance pattern j. After all disturbance patterns have been applied to the closed-loop system, the maximum of all resulting numbers is chosen as the value of $J_0(\mathbf{D})$.

The addition of input term I_{ij} in the objective function results in $\tilde{\mathcal{D}} = \varnothing$. This is because the definition of I_{ij} implies that $I_{ij} > 0$ for all non-trivial controllers. Therefore, provided $\lambda_{ij} > 0$, objective function J_0 will be non-zero for all non-trivial controllers. This means that no strictly optimal solutions can exist in $\mathcal{D}$. A direct consequence of this, is that a unique optimal solution should normally be expected when minimising J_0. Although the addition of I_{ij} in the objective function removes the useful properties associated with strictly optimal solutions, the improved robustness achieved by the inclusion of I_{ij} was judged as necessary, considering the difficulty of the Shell standard control problem. This approach is also very common in other optimisation-based control schemes, such as most predictive control algorithms (García and Morshedi, 1986; Clarke, Mohtadi, and Tuffs, 1987a, 1987b; García, Prett, and Morari, 1989; Clarke and Mohtadi, 1989).

6.4.2 Application to the Shell Standard Control Problem

In the case of the Shell standard control problem, the set of all permissible controller parameter vectors is $\mathcal{D} = \left\{ \left[K_{P_1}\ T_{I_1}\ T_{D_1}\ K_{P_2}\ T_{I_2}\ T_{D_2} \right] \in \mathbb{R}^6 \right\}$. Two disturbance test patterns were used for the closed-loop tests. Specifically, $\mathbf{d}_1 = [0.5\ 0.5]^T$ and $\mathbf{d}_2 = [-0.5\ -0.5]^T$ which represent the worst-case scenarios, since d_1 and d_2 are at the extremes of ± 0.5 and have the same sign. With reference to Eq. (6.19), $i = 1, 2$ (for the two PID loops), and $j = 1, 2$ (for the two disturbance patterns).

In the steady state, y_1 and y_2 must be maintained at 0.0 ± 0.005. In addition to that, y_1 must be maintained between 0.5 and –0.5 at all t, while there is no transient requirement

for y_2. These specifications were expressed as time-domain bounds by means of $f_{ij}^{(u)}(t)$ and $f_{ij}^{(l)}(t)$ which are shown in Fig. 6.10. They were kept constant for all disturbance test patterns. The maximum simulation time was set to be $t_{max} = 400$ minutes. The required settling time, as defined in (6.5), was set to be $t_s = 200$ minutes, which is well within the closed-loop bandwidth requirement. The weighting and scaling coefficients were set to be $\alpha_{ij} = \lambda_{ij} = 1$ and $\delta_i = 0.05T = 0.25$, respectively, for all i, j. In order to evaluate J_0, the closed-loop system shown in Fig. 6.8 was simulated in MATLAB/SIMULINK using the Runge-Kutta fifth-order numerical integration algorithm, with a constant step size of 1 minute. The integral in Eq. (6.17) was evaluated using the Euler method.

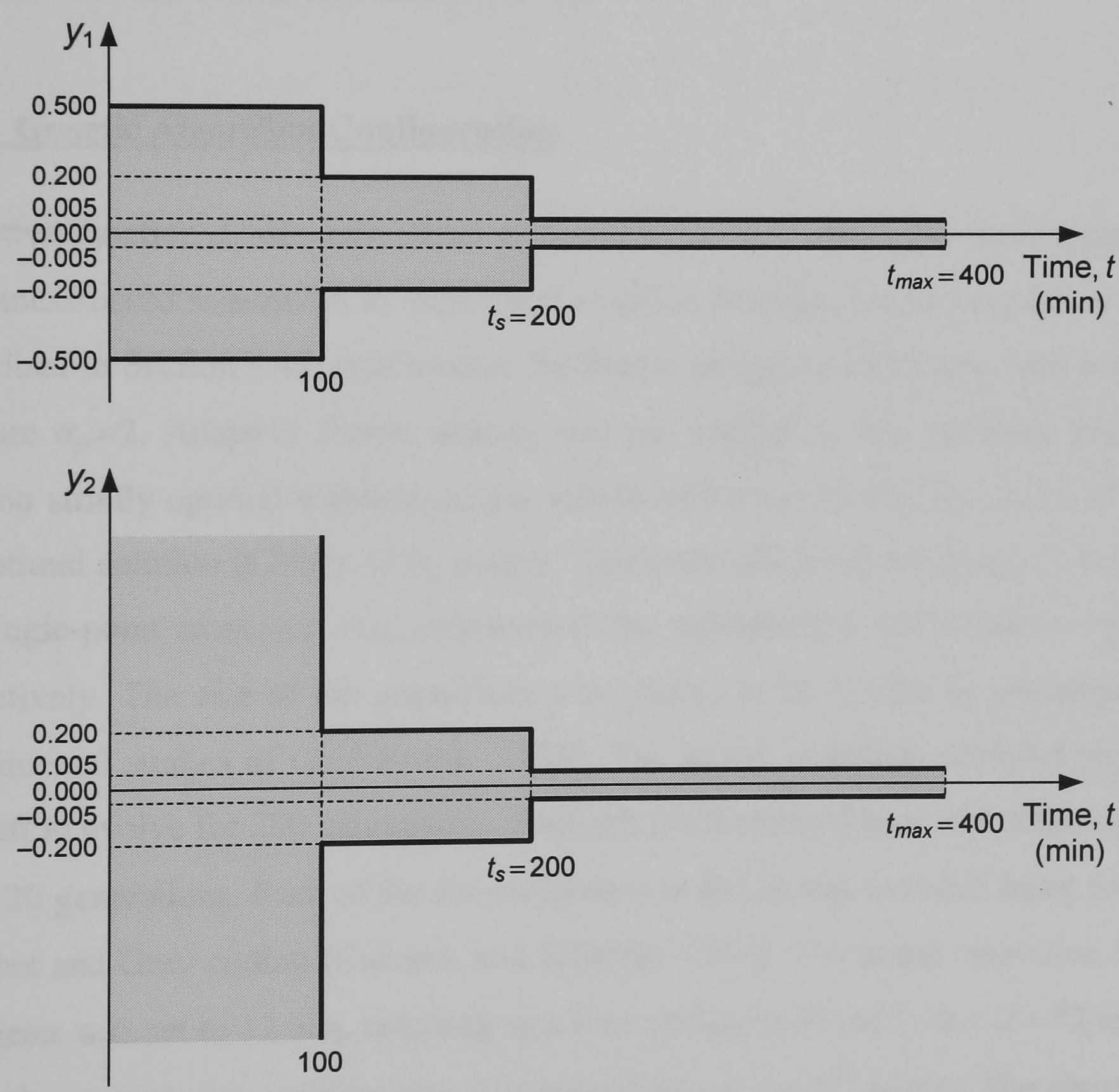


Fig. 6.10 *Performance specifications for outputs y_1 and y_2*

The transient specifications for output y_7 were not included in the objective function J_0, because the two PID controllers do not directly affect the value of y_7, and no feedback information about y_7 is used in the computation of controller outputs u_1 and u_2. The LP

approach used to compute u_3 guarantees that the constraint $y_7 \geq -0.5$ will not be violated in the steady state. However, constraint violations may occur in the transient period. To minimise these constraint violations on y_7, the optimal value of u_3 *at each sampling instant* must be computed, which implies solving the full minimisation problem in (6.5). This is a difficult task and was not attempted in this work. Cuthrell, Rivera, Schmidt, and Vegeais (1990) attempted to numerically solve (6.5) using QDMC and a non-linear programming-based controller, where it was observed that constraint violations on y_1, y_2 and y_7 still occurred in the transient period. This is a strong indication that a feasible solution to problem (6.5) may not exist at all t, and constraint violations in the transient period may thus be inevitable. An interesting discussion on the feasibility of (6.5) can be found in Prett, García, and Morari (1990).

6.4.3 Genetic Algorithm Configuration

Objective function J_0 was minimised using a simple GA, with a generation gap $g=0.9$ and fitness-based reinsertion to implement an elitist strategy. Linear population ranking (described in Section 4.42) was used as the fitness assignment strategy, with a selective pressure $\sigma_p=2$. Adaptive fitness sharing was not applied in this problem, since there exist no strictly optimal solutions in the search landscape (by the definition of J_0) and the optimal solution is likely to be unique. Stochastic universal sampling (Baker, 1987) and single-point crossover were employed in the reproduction and crossover operators, respectively. The size of the population was chosen to be $N=80$, in accordance with experimental studies of Grefenstette (1986). The initial, randomly selected population was left to evolve for 200 generations, although good convergence was achieved in less than 120 generations. Each of the six parameters in $\mathbf{D} \in \mathcal{D}$ was encoded using the binary alphabet and Gray coding (Caruana and Schaffer, 1988). The string resolution for each parameter was set to 12 bits, resulting in a total string length of $l=6\times12=72$ bits. This results in a search space whose size is approximately 4.7×10^{21} points. The chromosome structure used is similar to the one shown in Fig. 5.4. The crossover and mutation probabilities were chosen to be $p_c=0.45$ and $p_m=0.01$, respectively, using the guidelines of Grefenstette (1986). The PID controller proportional terms K_{P_i} were assumed to take values in the interval $[0,10]$, the integral terms T_{I_i} in the interval $[0.1,100]$ minutes, and the derivative terms T_{D_i} in the interval $[0,20]$ minutes.

6.5 Simulation Results

In order to test the performance of the control scheme, the closed-loop system was subjected to disturbance patterns $\mathbf{d}_1 = [0.5\ 0.5]^T$, $\mathbf{d}_2 = [-0.5\ -0.5]^T$, $\mathbf{d}_3 = [0.5\ -0.5]^T$, and $\mathbf{d}_4 = [-0.5\ 0.5]^T$, covering disturbances with the same sign and with opposite signs. Note that patterns $\mathbf{d}_3$ and $\mathbf{d}_4$ were not used in the PID tuning closed-loop tests. Figures 6.11 to 6.14 show the closed-loop responses obtained using the best PID controller parameters at generation 200. The regulation performance specifications for y_1 and y_2 (shown in Fig. 6.10), as well as the saturation constraint boundaries for u_1, u_2, and u_3 are also plotted for comparison.

It can be seen that, under all four disturbance test patterns, the closed-loop system has completely met the steady-state specifications. It is observed that outputs y_1 and y_2 are rapidly stabilised to zero while input u_3 is minimised. Furthermore, all manipulated variables are within the saturation and rate limit constraints. It is also observed that when $\mathbf{d} = \mathbf{d}_1$ (Fig. 6.11), y_7 settles precisely at its constraint boundary of –0.5. This means that the steady-state value of u_3 is indeed the lowest possible value such that all objectives are satisfied and no constraints are violated in the steady state. If u_3 is set to a lower value, y_7 will drop below –0.5 causing a constraint violation. This situation also occurs when $\mathbf{d} = \mathbf{d}_4$ (Fig. 6.14). Similarly, when $\mathbf{d} = \mathbf{d}_2$ (Fig. 6.12), it is u_1 which now settles precisely at its upper constraint boundary of 0.5, indicating that the steady-state value of u_3 is again optimal. If u_3 is set to a lower value, u_1 will have to be increased to a value greater than 0.5 to satisfy all other requirements. But u_1 is already constrained such that $|u_1| \leq 0.5$. This will cause the actuator of u_1 to saturate, resulting in a negative offset in y_1, thus dissatisfying the regulation requirement for y_1. This situation also occurs when $\mathbf{d} = \mathbf{d}_3$ (Fig. 6.13).

In terms of transient specifications, it is observed that small constraint violations occur for y_1 in the time range between 45 and 70 minutes (Figures 6.11 and 6.12). Transient constraint violations also occur for y_7 in the first 220 minutes. Setting $\lambda_{ij} = 0$ for all i, j and repeating the PID tuning did not solve the problem. Note, however, that similar violations also occur when using the QDMC algorithm, as well as several other modern approaches (see Prett, García, and Morari, 1990). It is, therefore, conjectured that the transient response output constraint violations arise necessarily from the structure of the controlled process, and thus cannot be avoided.

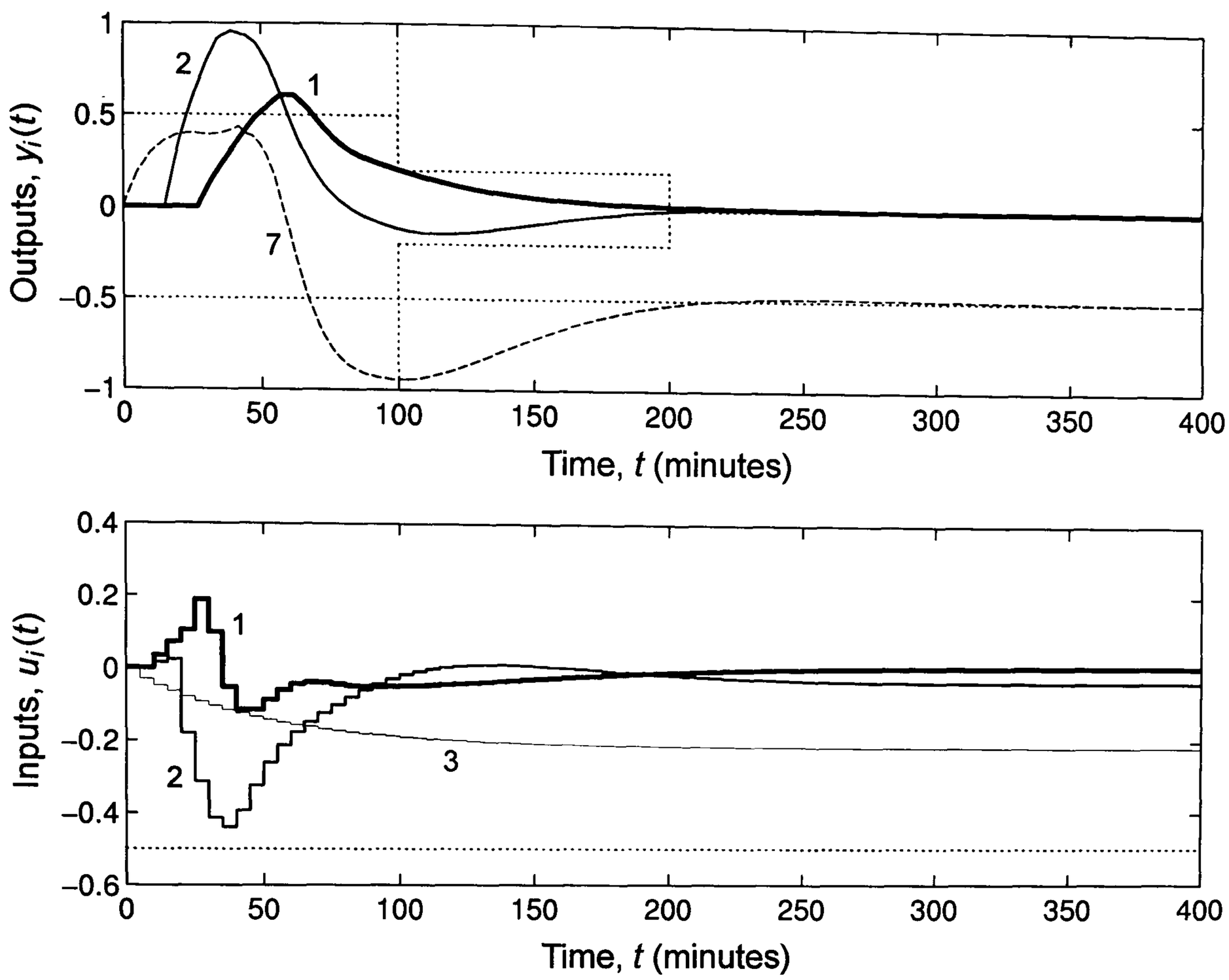


Fig. 6.11 *Closed-loop system responses under disturbance pattern* $\mathbf{d}_1 = [0.5\ 0.5]^T$

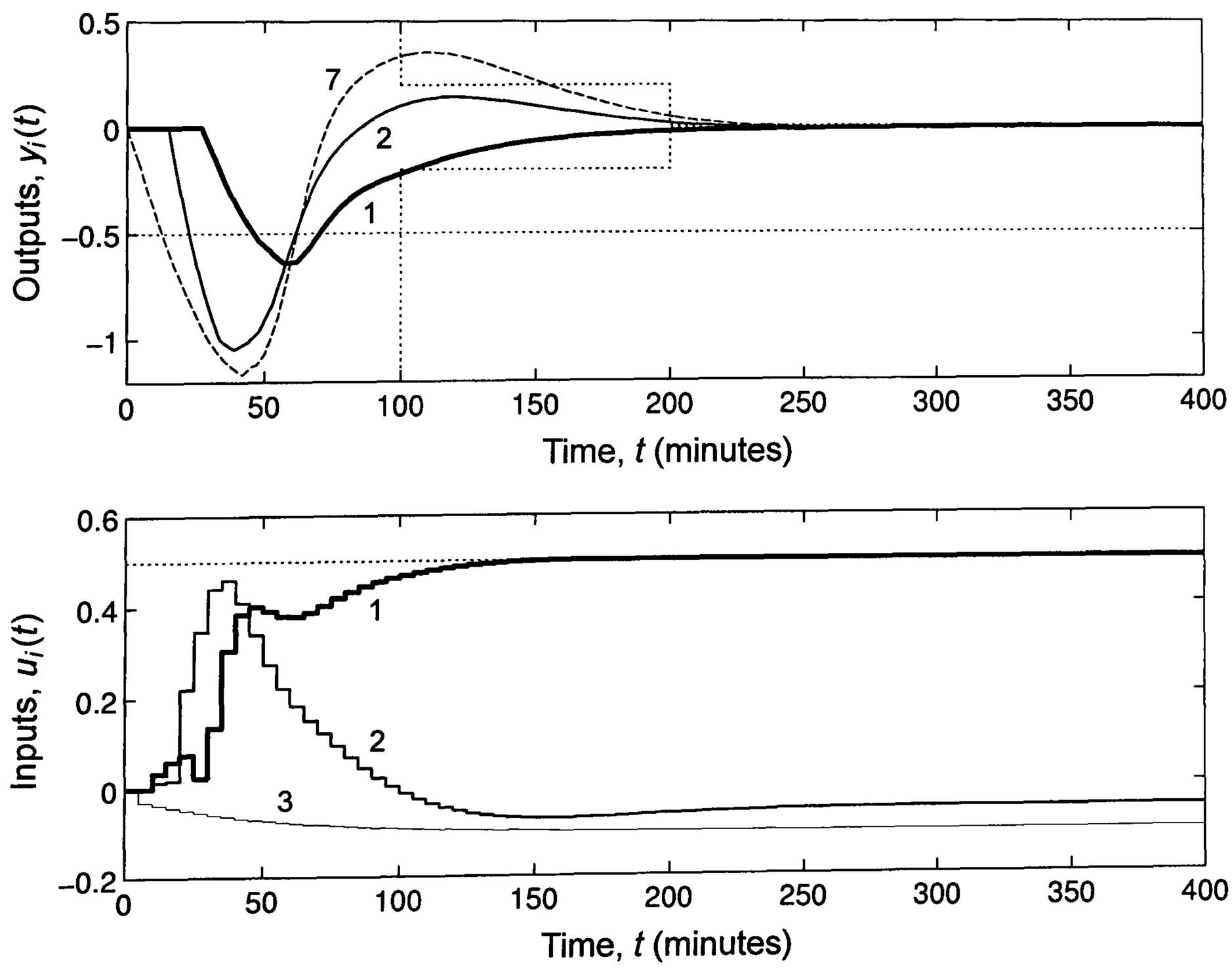


Fig. 6.12 *Closed-loop system responses under disturbance pattern* $\mathbf{d}_2 = [-0.5\ -0.5]^T$

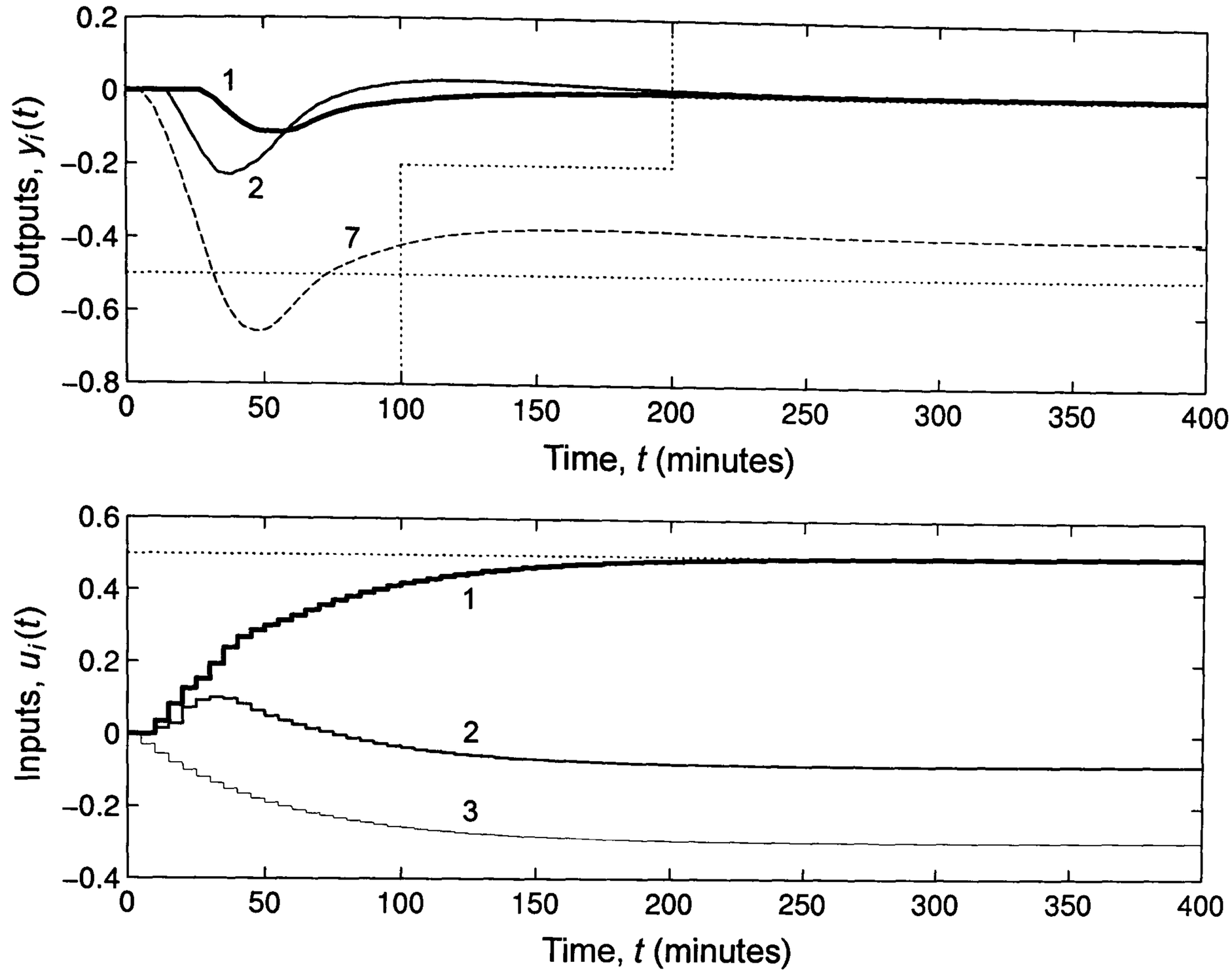


Fig. 6.13 *Closed-loop system responses under disturbance pattern* $\mathbf{d}_3 = [0.5\ -0.5]^T$

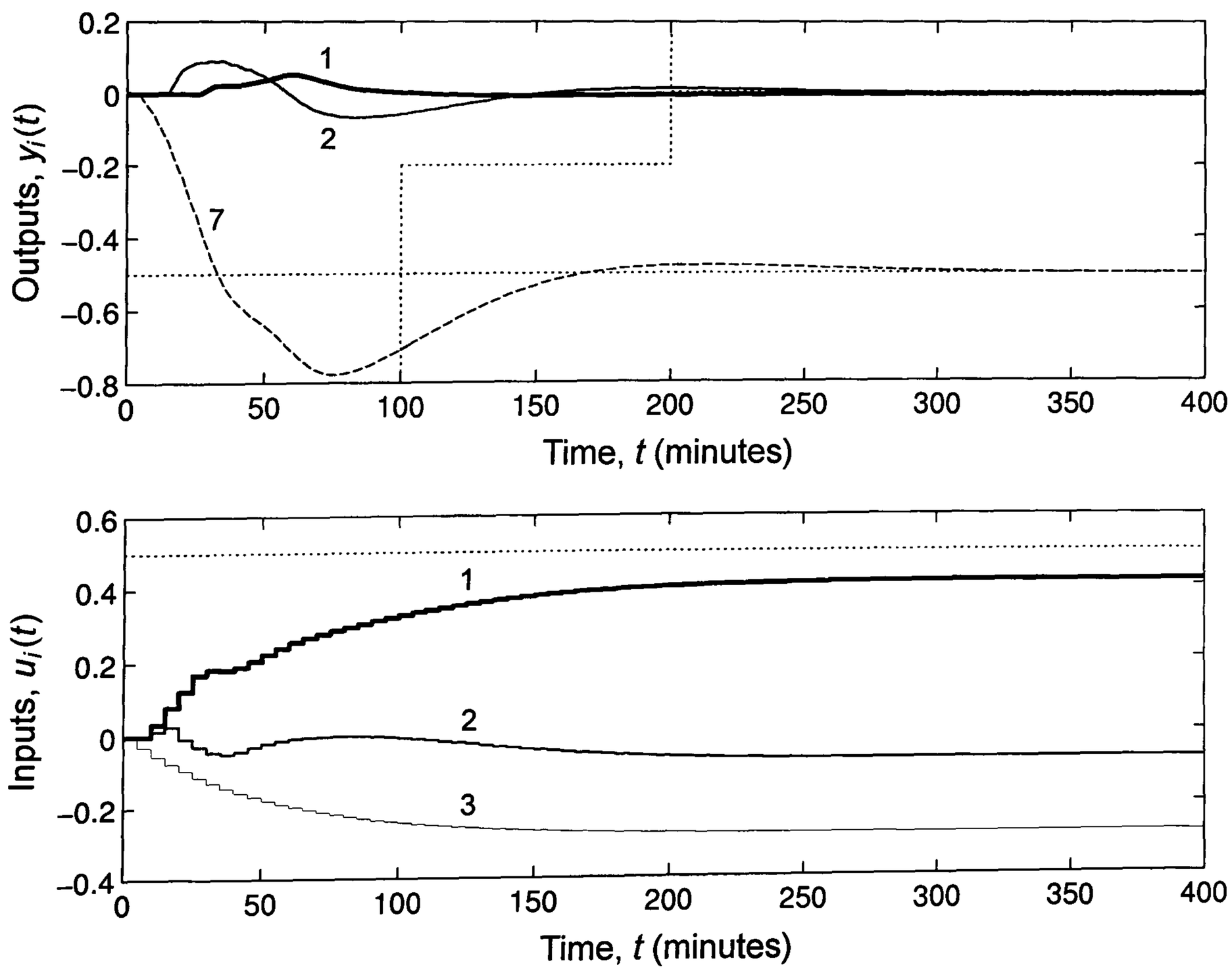


Fig. 6.14 *Closed-loop system responses under disturbance pattern* $\mathbf{d}_4 = [-0.5\ 0.5]^T$

The parameters of the optimal controller, $\mathbf{D}_{opt}$, obtained by the GA, as well as the value of $J_0(\mathbf{D}_{opt})$, are shown in Table 6.2 below.

TABLE 6.2 Optimal PID controller parameters with $\alpha_{ij}=1$ and $\lambda_{ij}=1$ for all i, j

Parameter	PID Controller 1	PID Controller 2
Proportional gain, K_{P_i}	1.5043	0.5690
Integral action time, T_{I_i} (minutes)	7.1259	39.8404
Derivative action time, T_{D_i} (minutes)	0	0
Objective function, J_0	4.9631	

It is observed that $K_{P_1} \approx 3K_{P_2}$ and $T_{I_2} \approx 6T_{I_1}$, indicating that loop 1 requires tighter control. This is mainly because of the additional transient requirement for y_1 which does not exist for y_2 (see Fig. 6.10). An interesting observation is that $T_{D_1} = T_{D_2} = 0$, indicating that the derivative terms in both loops have been switched off. This is not surprising since PI control is generally sufficient for processes with first-order dynamics, such as the available model of the Shell process.

In order to see the effect of the term I_{ij} in the objective function J_0 in Eq. (6.19), the GA run was repeated with $\lambda_{ij}=0$. The parameters of the optimal controller, $\mathbf{D}_{opt}$, obtained by the GA, as well as the value of $J_0(\mathbf{D}_{opt})$, are shown in Table 6.3 below.

TABLE 6.3 Optimal PID controller parameters with $\alpha_{ij}=1$ and $\lambda_{ij}=0$ for all i, j

Parameter	PID Controller 1	PID Controller 2
Proportional gain, K_{P_i}	2.8327	1.1624
Integral action time, T_{I_i} (minutes)	4.4912	43.2558
Derivative action time, T_{D_i} (minutes)	0	3.8974
Objective function, J_0	1.8618	

It is observed that $\lambda_{ij}=0$ for all i, j resulted in controllers with higher proportional gains, a lower integral action time for the controller in loop 1, and a non-zero derivative action time for the controller in loop 2. This was expected since $\lambda_{ij}=0$ means that term I_{ij}, which penalises the activity of the manipulated variables, is not used in the objective

function. In terms of transient output constraint violations, the PID controllers obtained achieved marginally better performance than those in Table 6.2, but at the expense of unwanted oscillatory behaviour in the manipulated variables, as can be observed in Fig. 6.15 where the closed-loop system responses under disturbance $\mathbf{d}_1$ are shown. This is a strong indication of poor robustness. It is, therefore, concluded that term I_{ij} is an important element of J_0, which can improve the robustness of the closed-loop system. A desirable balance between performance and robustness can be achieved by appropriately adjusting the weighting coefficients α_{ij} and λ_{ij}.

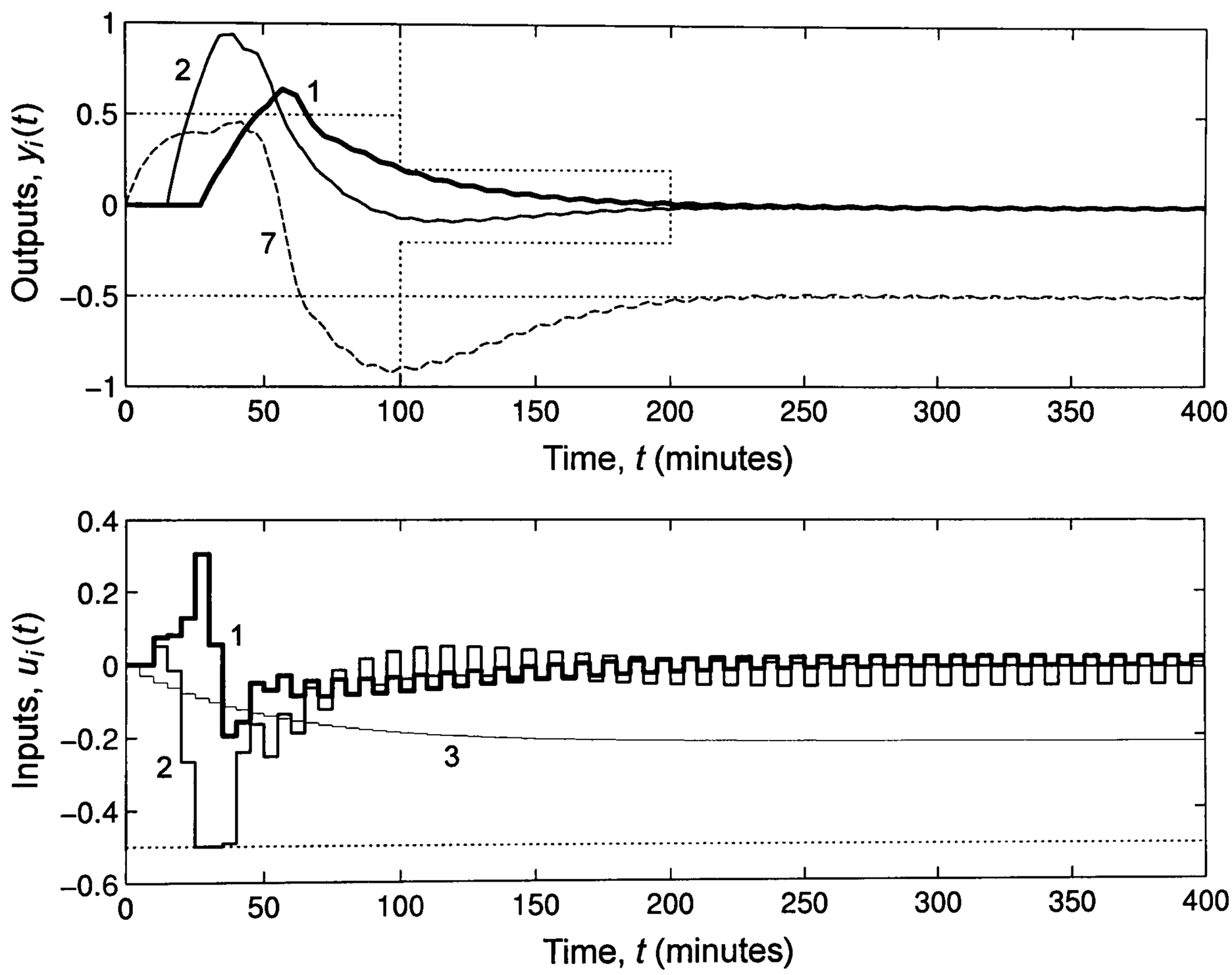


Fig. 6.15 *Closed-loop system responses under disturbance pattern* $\mathbf{d}_1 = [0.5\ 0.5]^T$ *using the PID controller parameters given in Table 6.3*

6.5.1 Measurement Noise and Disturbance Variations

The genetically tuned PID controllers were also tested under noisy output measurements and disturbance variations. Specifically, measurement noise was added to all measured process outputs, y_1, y_2, y_3, y_5, y_7. The noise signals used were independent, normally distributed sequences, all having zero mean and variance $\sigma_m^2 = 1 \times 10^{-3}$. The closed-loop

system was subjected to the four disturbance patterns $\mathbf{d}_1 = [0.5\ 0.5]^T$, $\mathbf{d}_2 = [-0.5\ -0.5]^T$, $\mathbf{d}_3 = [0.5\ -0.5]^T$, and $\mathbf{d}_4 = [-0.5\ 0.5]^T$, which were applied in the following sequence.

$$\mathbf{d}(t) = \begin{cases} \mathbf{d}_1(t) = [0.5\ 0.5]^T, & 0 \le t < 400 \\ \mathbf{d}_2(t) = [-0.5\ -0.5]^T, & 400 \le t < 800 \\ \mathbf{d}_3(t) = [0.5\ -0.5]^T, & 800 \le t < 1200 \\ \mathbf{d}_4(t) = [-0.5\ 0.5]^T, & 1200 \le t \le 1600 \end{cases} \tag{6.20}$$

where t is in minutes. The closed-loop responses without measurement noise are shown in Fig. 6.16, while those with measurement noise are shown in Fig. 6.17. It is observed that the presence of measurement noise does not significantly affect the performance of the closed-loop system. As expected, the disturbance vector transition from $\mathbf{d}_1$ to $\mathbf{d}_2$ at $t = 400$ minutes results in large deviations from the transient specifications for outputs y_1, y_2, and y_7. Note, however, that the Shell standard control problem specifications assume that all signals are at zero prior to a disturbance change (zero initial conditions). Since $|d_1| \le 0.5$ and $|d_2| \le 0.5$, this results in disturbance changes with $\|\Delta\mathbf{d}\| \le \sqrt{2}/2$. In function $\mathbf{d}(t)$ in (6.20), the transition at $t = 400$ minutes is of magnitude $\|\mathbf{d}_2 - \mathbf{d}_1\| = \sqrt{2}$ which is twice the maximum $\|\Delta\mathbf{d}\|$. This is the reason for the large constraint violations that are observed after the disturbance change at $t = 400$ minutes. In the noise-free case (Fig. 6.16), the disturbance estimates accurately track the real disturbances, with a lag of one sample as theoretically predicted earlier. These estimates are then used by the LP solver, which accurately computes the optimal steady-state value of u_3. In the case where measurement noise was added to the system (Fig. 6.17), it is observed that the output responses are qualitatively very similar to those in the noise-free case, indicating good noise immunity. It can be seen that the noise propagates through to the disturbance estimates, and also to the LP solution which is filtered by a low-pass filter before being applied to the process. In order to reject the noise in the disturbance estimator outputs, filter $\mathbf{G}_F(z)$ in Eq. (6.15) was also used. $\mathbf{G}_F(z)$ consists of two filters, both with a time constant of 30 minutes. The obtained results are shown in Fig. 6.18.

$$\mathbf{G}_F(z) = \begin{bmatrix} \dfrac{0.1535z}{z-0.8465} & 0 \\ 0 & \dfrac{0.1535z}{z-0.8465} \end{bmatrix} \tag{6.21}$$

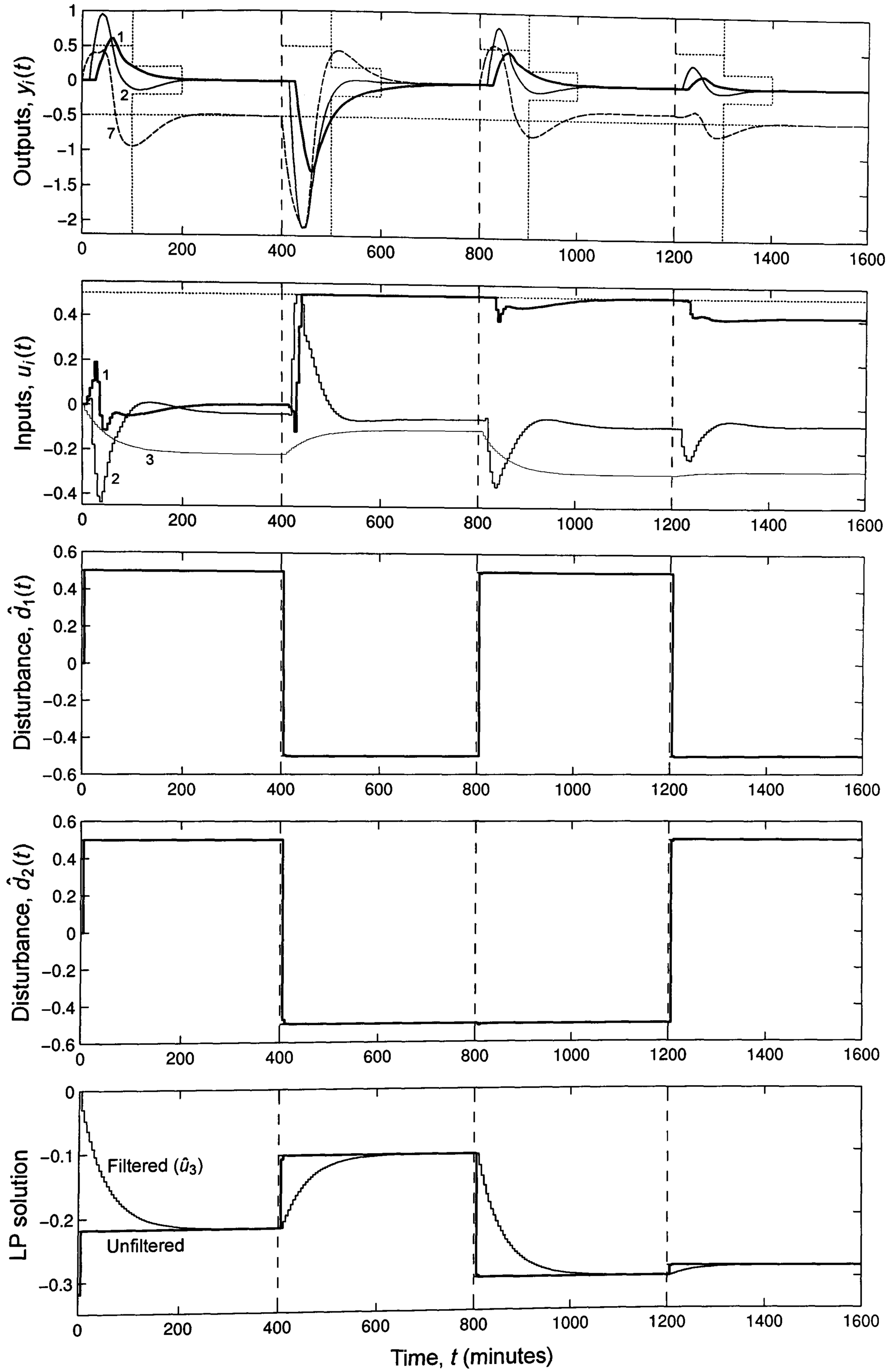


Fig. 6.16 *Closed-loop system responses without measurement noise*

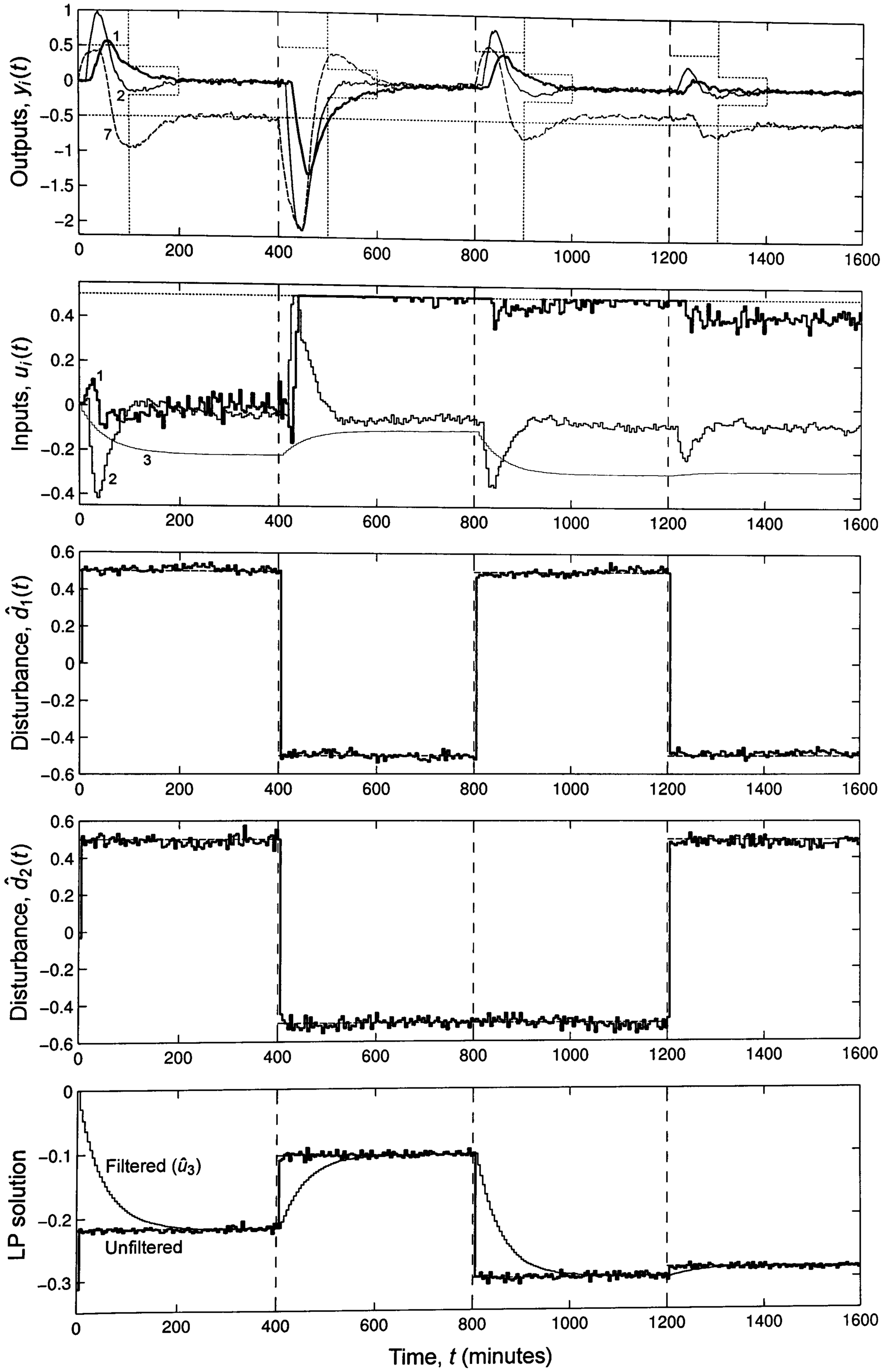


Fig. 6.17 Closed-loop system responses with measurement noise

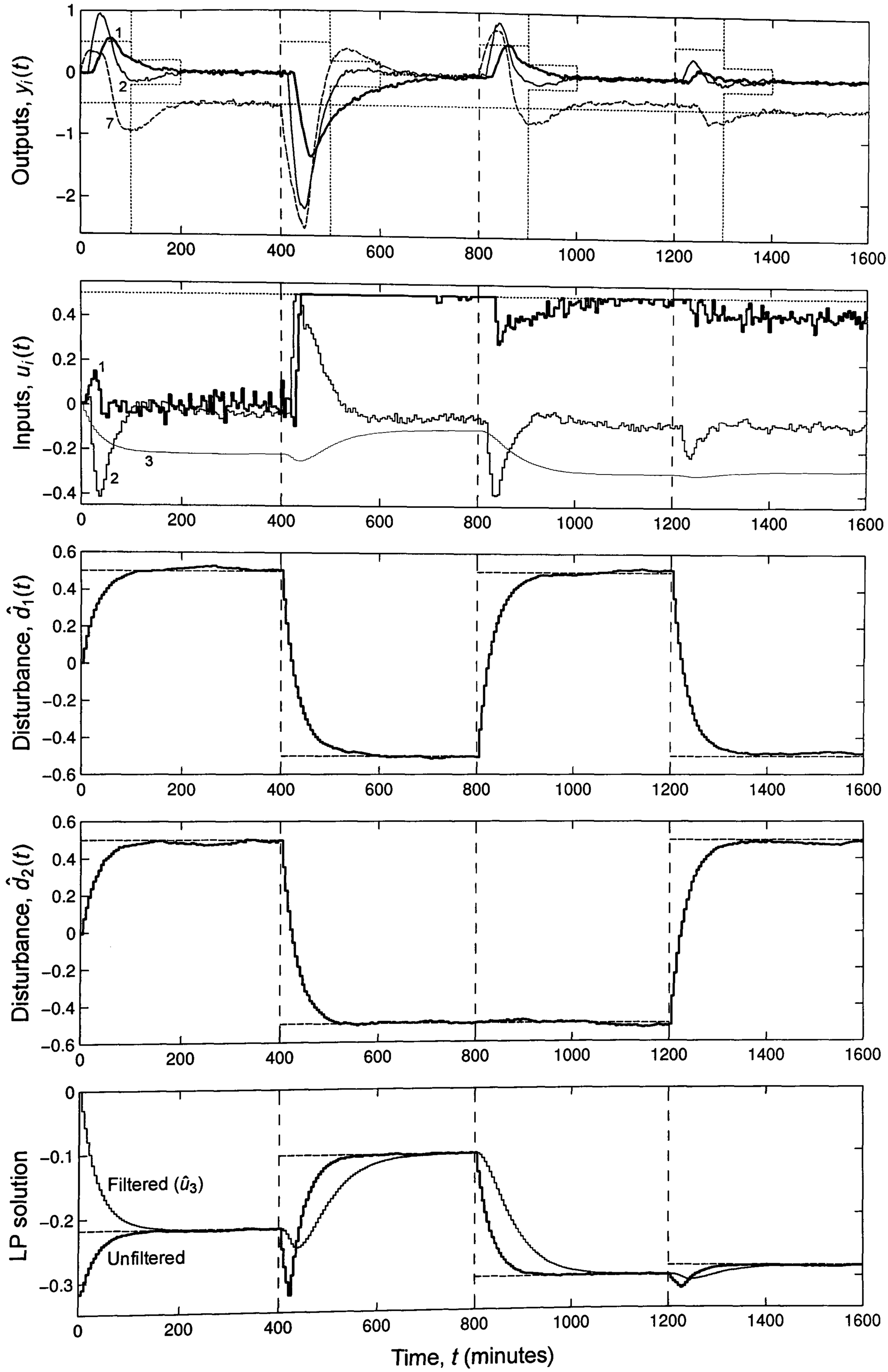


Fig. 6.18 *Closed-loop system responses with measurement noise and disturbance filtering*

It is observed from Fig. 6.18 that filter $\mathbf{G}_F(z)$ in the disturbance estimator results in much less noisy disturbance estimates, but at the expense of a much slower disturbance tracking time. Although the LP solver output is now less noisy than that in Fig. 6.17, it is observed that there are large undershoots around 400 and 1200 minutes. The reason for this behaviour is that the LP solver output is a *non-linear* function of the disturbance estimates, which means that slow changes in the values of the disturbance estimates in the direction of the real disturbances does not necessarily imply slow changes in the LP solver output in the direction of its optimal value. With reference to Fig. 6.18, although the optimal steady-state values of u_3 when $\mathbf{d}=\mathbf{d}_1$ and $\mathbf{d}=\mathbf{d}_2$ are –0.2178 and –0.1040, respectively (as observed in Table 6.1), the slow tracking of the real disturbance vector results in $\hat{\mathbf{d}} \approx [0\ 0]^T$ at $t=420$ minutes. The optimal steady-state value of u_3 in this case is –0.3181 (because of the non-linearity of the LP solver). This causes the undershoot in the LP solver output at $t=420$ minutes. A similar situation occurs at $t=1220$ minutes. These undershoots reduce the overall performance of the closed-loop system, as can be observed by comparing the output responses of Fig. 6.18 with those of Fig. 6.17. It is, therefore, concluded that filter $\mathbf{G}_F(z)$ should be used with caution when combined with non-linear elements such as the LP solver used in this work. $\mathbf{G}_F(z)$ may, however, be helpful in cases where the unfiltered $\hat{\mathbf{d}}$ is heavily corrupted with noise.

6.5.2 Robust Stability and Performance Tests

In all simulation results presented earlier in this section, it was assumed that there is no uncertainty in the gains of the process model. However, in the Shell standard control problem statement, the gains of all 35 transfer functions of the process model $\mathbf{G}(s)$ are subject to uncertainties of the following form.

$$\mathbf{G}(s) = \begin{bmatrix} \dfrac{K_{11}+\varepsilon_1\Delta K_{11}}{T_{11}s+1}e^{-sL_{11}} & \cdots & \dfrac{K_{15}+\varepsilon_5\Delta K_{15}}{T_{15}s+1}e^{-sL_{15}} \\ \vdots & \ddots & \vdots \\ \dfrac{K_{71}+\varepsilon_1\Delta K_{71}}{T_{71}s+1}e^{-sL_{71}} & \cdots & \dfrac{K_{75}+\varepsilon_5\Delta K_{75}}{T_{75}s+1}e^{-sL_{75}} \end{bmatrix} \tag{6.22}$$

where K_{ij}, T_{ij}, and L_{ij} denote the nominal gain, time constant, and time delay of the i,j element of $\mathbf{G}(s)$, ΔK_{ij} denotes the absolute maximum uncertainty of gain K_{ij}, and ε_j

determines the amount of uncertainty of gain K_{ij}, with $-1 \leq \varepsilon_j \leq 1$ and $j = 1, \ldots, 5$. The numerical values for K_{ij}, T_{ij}, L_{ij}, and ΔK_{ij} can be found in Appendix A. According to the Shell standard control problem statement (see Appendix A), the closed-loop system should satisfy all control objectives for all the plants in the uncertainty set (i.e. for all combinations of ε_j). The Shell problem statement contains a number of prototype test cases, which have been formulated in order to provide a common frame of reference for evaluating the robust stability and performance of different design approaches. These prototype test cases are given in Table 6.4 below.

TABLE 6.4 Prototype test cases for the Shell standard control problem (Prett and Morari, 1987)

Test case	ε_1	ε_2	ε_3	ε_4	ε_5	d_1	d_2	Optimal value of u_3
1	0	0	0	0	0	+0.5	+0.5	–0.2178
2	–1	–1	–1	+1	+1	–0.5	–0.5	+0.0782
3	+1	–1	+1	+1	+1	–0.5	–0.5	–0.2146
4	+1	+1	+1	+1	+1	–0.5	+0.5	–0.2494
5	–1	+1	0	0	0	–0.5	–0.5	+0.0386

Prototype test case 1 corresponds to the nominal process (since $\varepsilon_j = 0$ for all $j = 1, \ldots, 5$), with $\mathbf{d} = [0.5\ 0.5]^T$. This case has already been investigated earlier in Section 6.5, with the closed-loop responses shown in Fig. 6.11. In order to evaluate the robustness of the closed-loop system in terms of stability and performance, the remaining four prototype test cases were also investigated. All of these cases introduce large uncertainties in the gains of the process. The closed-loop responses for prototype test cases 2 to 5 are shown in Figures 6.19 to 6.22, respectively. The corresponding optimal steady-state values of u_3 for the four uncertain processes, shown in Table 6.4, are also shown in the figures for comparison. The results for test cases 2, 3, and 5 should be compared with those shown in Fig. 6.12 because the latter uses the same disturbance vector $\mathbf{d} = [-0.5\ -0.5]^T$, but without uncertainty. Similarly, the results for test case 4 should be compared with those shown in Fig. 6.14 where $\mathbf{d} = [-0.5\ 0.5]^T$.

Although the proposed design was only based on the nominal process parameters, it is observed that all prototype test cases result in a stable and well-damped closed-loop system, which strongly suggests that the closed-loop system is robustly stable. In test case 2 (Fig. 6.19), it can be seen that y_1 and y_2 are stabilised to zero, but the transient

constraint violations are now larger than those in Fig. 6.12 (no uncertainty). It is also observed that u_3 now converges to a value that is slightly larger than the optimal. This is because the disturbance estimator and the LP solver used in the proposed scheme are both based on the nominal process parameters which are now significantly different than those of the real (uncertain) process. The sub-optimality of u_3 can also be inferred by the fact that none of the constrained variables, u_1, u_2, u_3, y_7, approaches a constraint boundary in the steady-state. However, the difference from the true optimal steady-state value of u_3 is not very large, considering the severity of the gain uncertainties present in the process (gains K_{11}, K_{21}, K_{31}, and K_{71}, for example, which are extensively used in the proposed scheme, all have values which are less than half of their nominal ones). In fact, the observed performance of the scheme proposed in this work appears to be better than that of a QDMC controller that is tuned for the nominal process, and is comparable to that of schemes in which the model gain uncertainty is considered *explicitly* (see, for example, Cuthrell, Rivera, Schmidt, and Vegeais, 1990, who proposed a solution to the Shell standard control problem based on a nominal process QDMC design, as well as a non-linear programming approach that explicitly considers model uncertainty).

In test case 3 (Fig. 6.20) it is observed that, although y_2 achieves the regulation objective and is stabilised to zero, there is a steady-state offset in y_1 which settles at a non-zero value, indicating that integral action in loop 1 cannot be achieved. This problem arises because of the steady-state value of u_3 computed by the LP solver, which is now *smaller* than the optimal and is thus *infeasible*. This causes u_1 to saturate heavily, thus removing the integral action from loop 1. Apart from the steady-state offset in y_1, the observed performance is qualitatively comparable to that of QDMC and other, more advanced schemes (see, for example, Cuthrell, Rivera, Schmidt, and Vegeais, 1990). It will be shown later in this section that the steady-state offset in y_1 can be completely eliminated by appropriately modifying the linear program in (6.7).

Finally, in both test cases 4 (Fig. 6.21) and 5 (Fig. 6.22), the proposed scheme achieves zero offset for both y_1 and y_2, with values for u_3 which, although sub-optimal, are close to the optimal ones. The output responses in test case 4 are more oscillatory than those shown in Fig. 6.14 (no uncertainty), but the output constraint violations are similar. In test case 5, the output constraint violations are now slightly larger that those shown in Fig. 6.12 (no uncertainty).

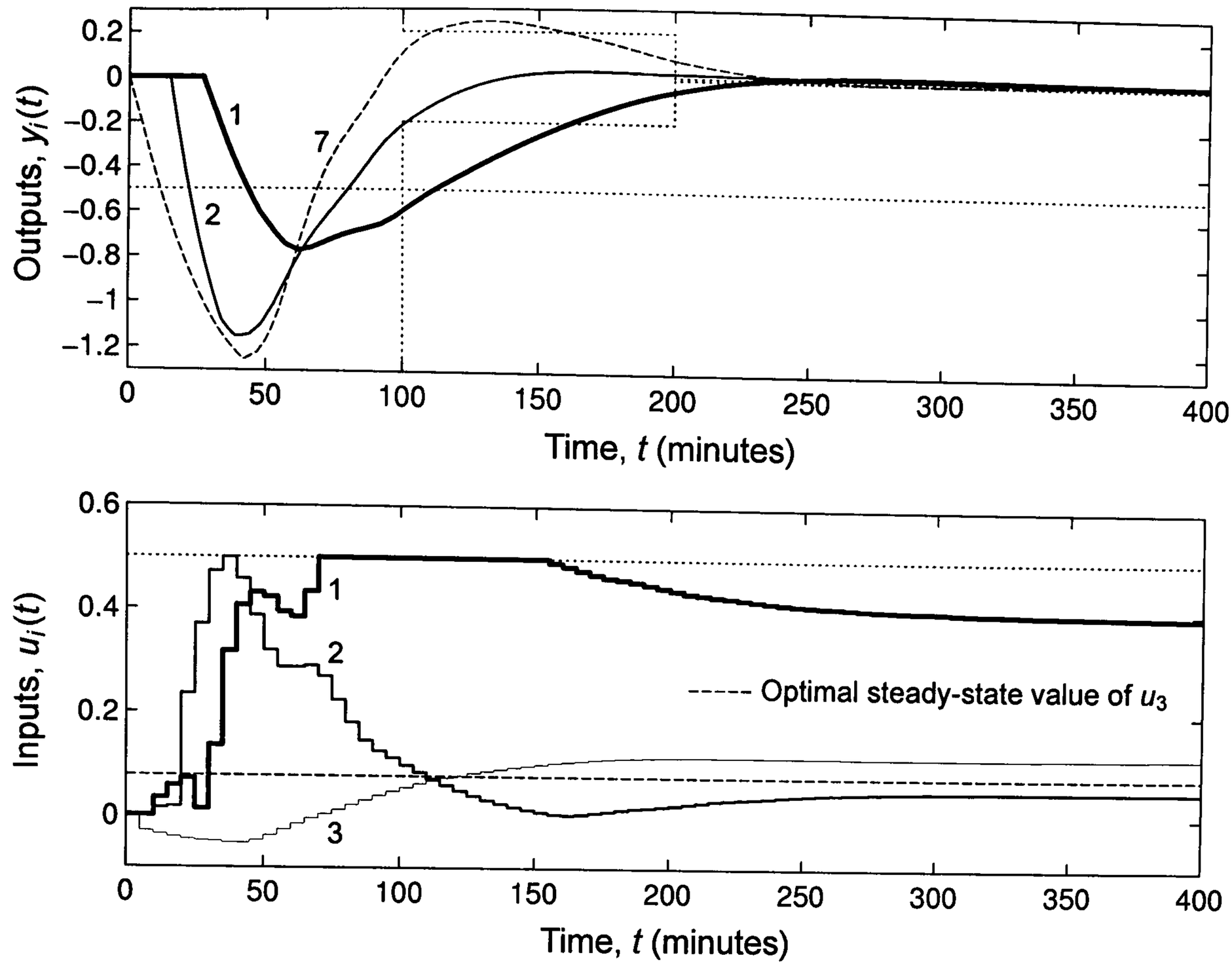


Fig. 6.19 *Closed-loop system responses for prototype test case 2*

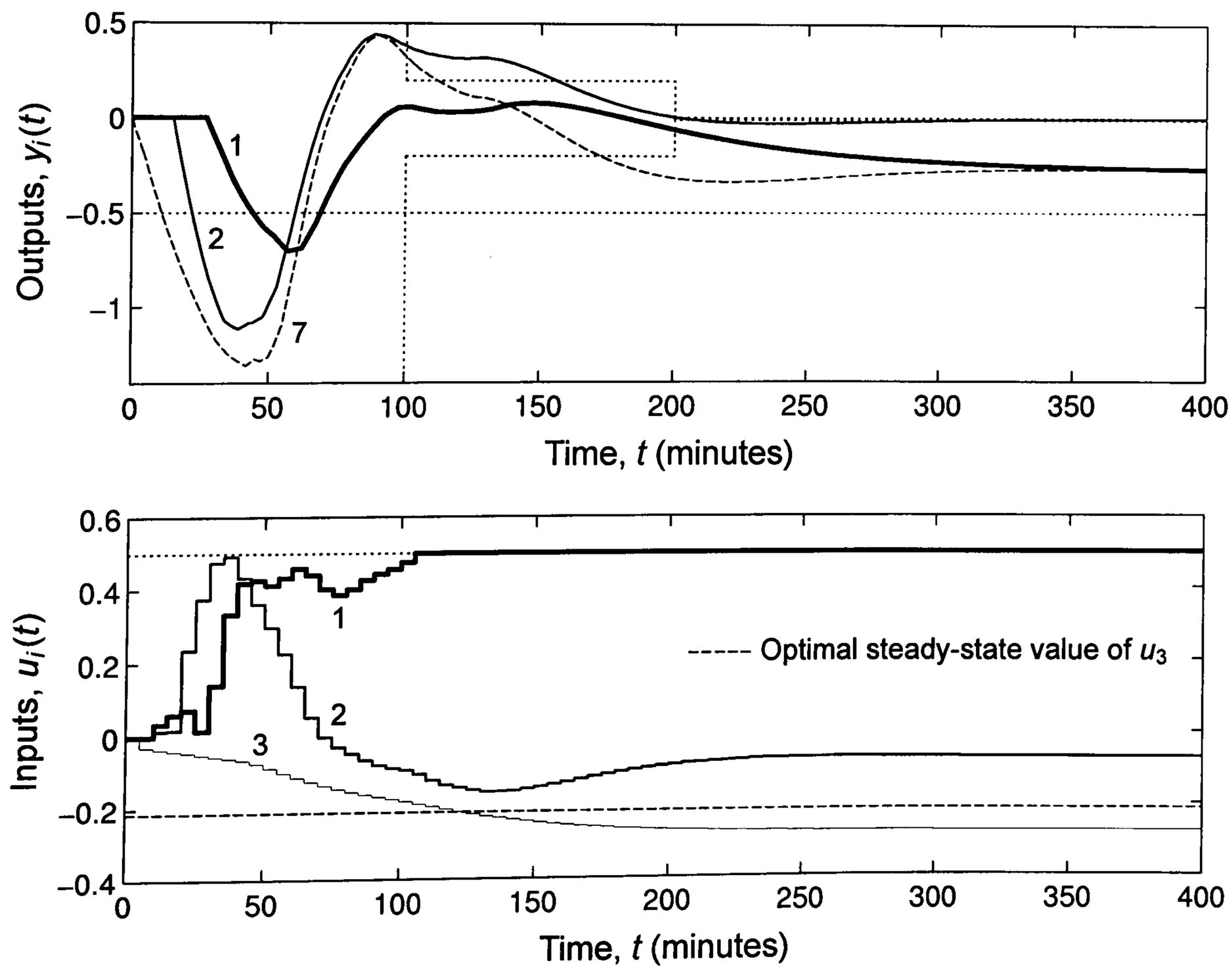


Fig. 6.20 *Closed-loop system responses for prototype test case 3*

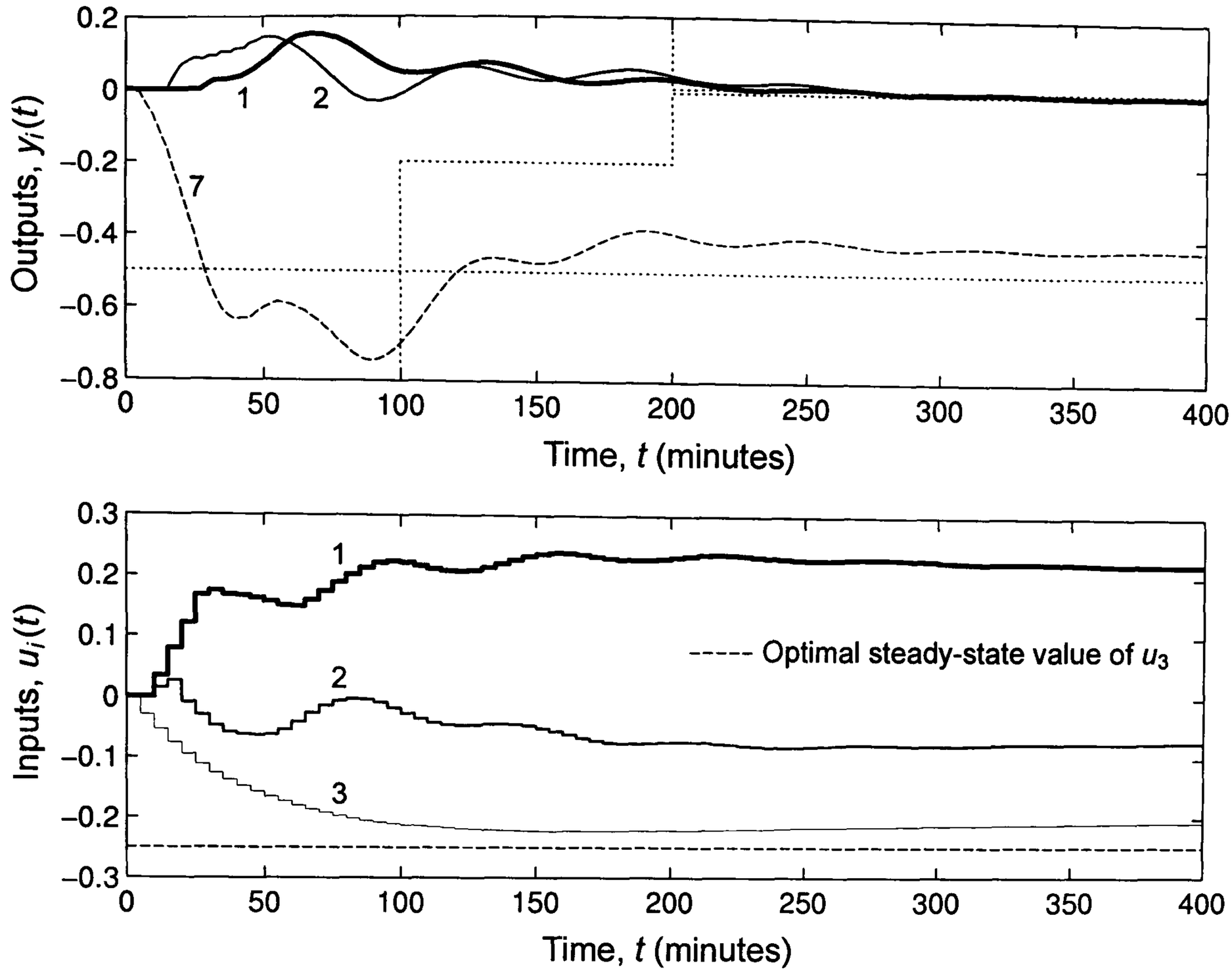


Fig. 6.21 *Closed-loop system responses for prototype test case 4*

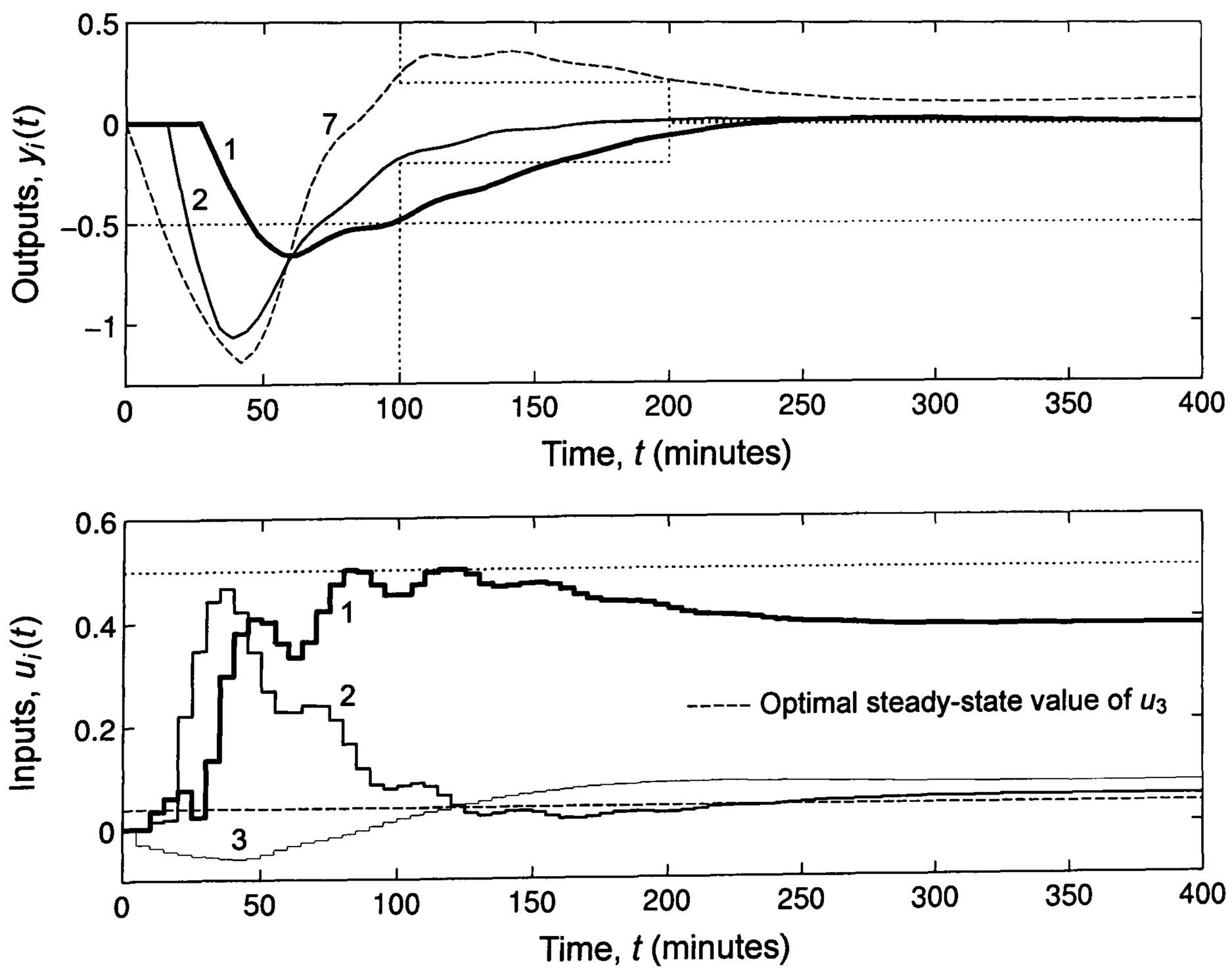


Fig. 6.22 *Closed-loop system responses for prototype test case 5*

It was shown in Section 6.3.2 that when $\varepsilon_j = 0$ for all $j = 1, \ldots, 5$ (no uncertainty), the LP solver used in the proposed scheme computes the steady-state value of u_3 that brings the process *at the verge of at least one constraint violation.* Also, neither the LP solver, nor the disturbance estimator consider the model gain uncertainties explicitly, but are only based on the nominal process model. This means that gain mismatches between the real process and the nominal process model will almost certainly cause the LP solver output to deviate from the optimal value of u_3. If the LP solver output satisfies all conditions of linear program (6.7) for the *real (not the nominal) process*, the obtained solution will be *sub-optimal* and may still be acceptable. This situation can be observed in prototype test cases 2, 4, and 5 (Figures 6.19, 6.21, and 6.22, respectively). On the other hand, if the LP solver output does not satisfy all conditions of linear program (6.7) for the real (not the nominal) process, the solution will be *infeasible* and, therefore, not acceptable. This situation can be observed in prototype test case 3 (Fig. 6.20), where the equality $y_1 = 0$ in linear program (6.7) is not satisfied, resulting in the steady-state offset in y_1.

In order to alleviate this problem and improve the robust performance of the proposed scheme, more conservative constraint boundaries may be specified in (6.7), so that the effects of uncertainties are accounted for. Consider the following linear program.

$$\min_{u_i} u_3, \quad i = 1, 2, 3$$

$$\text{Subject to:} \quad \begin{aligned} |u_i| &\leq 0.5\eta, \quad i = 1, 2, 3 \\ |y_i| &= 0, \quad i = 1, 2 \\ y_7 &\geq -0.5\eta \end{aligned} \tag{6.23}$$

where $0 \leq \eta \leq 1$. Setting $\eta = 1$ results in linear program (6.7), while the smaller the value of η, the more conservative the constraint boundaries. Setting $\eta < 1$ will inevitably result in the LP solver generating sub-optimal solutions for u_3, but these solutions will now be feasible on a wider range of processes within the uncertainty set. This is similar to the heuristic approach of de-tuning a controller in order to increase its robustness. Fig. 6.23 shows the closed-loop responses for prototype test case 3, with $\eta = 0.75$. It is observed that the steady-state offset in y_1 has now been completely eliminated, and the value of u_3 is marginally larger than the optimal.

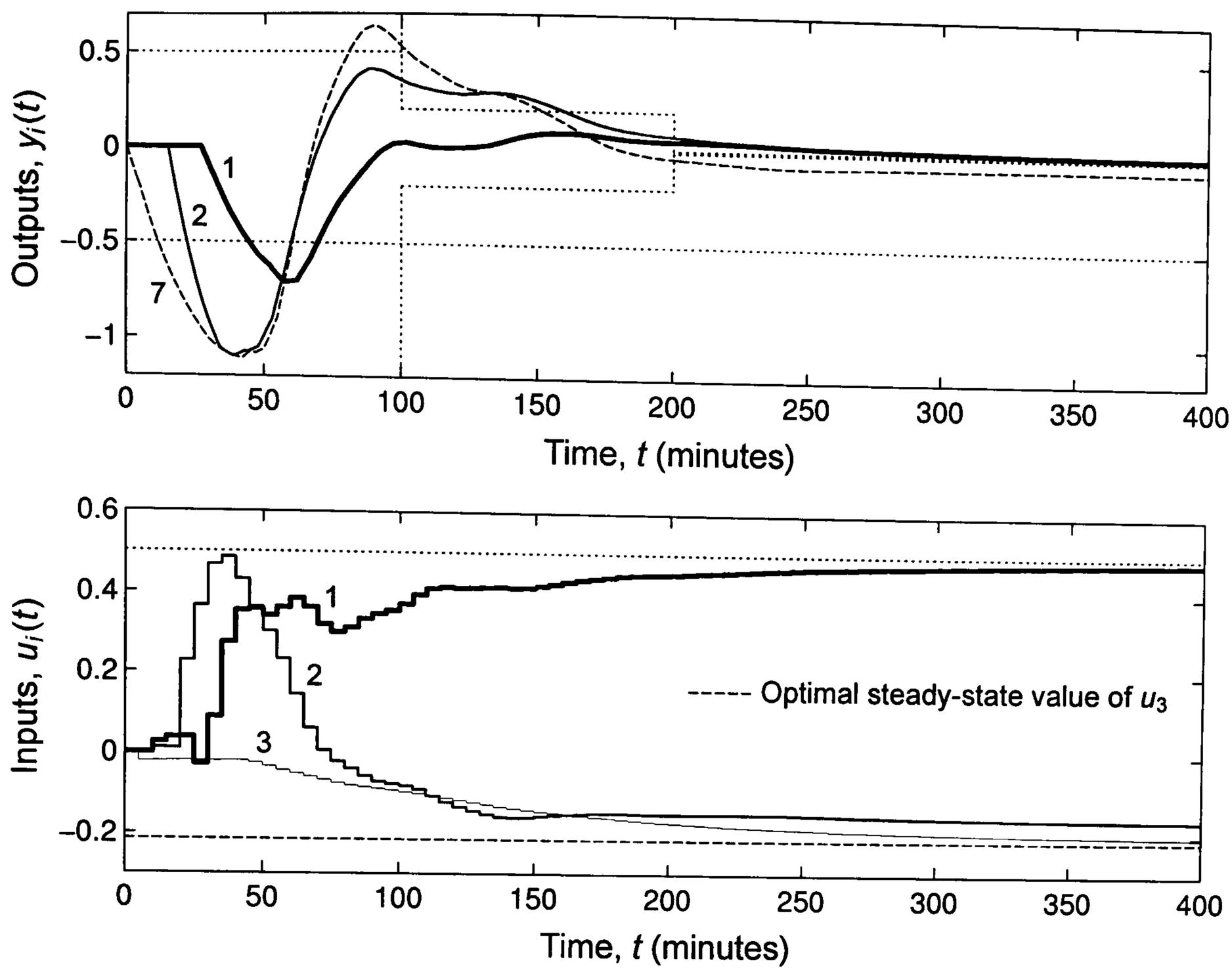


Fig. 6.23 Closed-loop system responses for prototype test case 3, with $\eta=0.75$

Prototype test cases 1, 2, 4, and 5 were also evaluated using $\eta=0.75$, in order to see how this setting affects the obtained results. It was clearly observed that in all cases the obtained responses were qualitatively similar to those obtained when $\eta=1$. As expected, when $\eta=0.75$ the LP solver output converged to values for u_3 slightly larger than those obtained when $\eta=1$. Interestingly, the transient constraint violations were fewer when $\eta=0.75$. This is because of the conservatism in the constraint boundaries used.

6.5.3 Genetic Algorithm Convergence

The optimisation performance of the GA can be observed in Fig. 6.24, which illustrates the convergence of the objective function J_0, as well as that of the proportional, integral, and derivative terms of the two PID controllers. Fig. 6.24 shows the parameters of the best pair of PID controllers in each generation. It can be seen that good convergence for all parameters is achieved in less than 120 generations. The solution shown in Table 6.2 was attained at generation 134, indicated by the vertical dashed line.

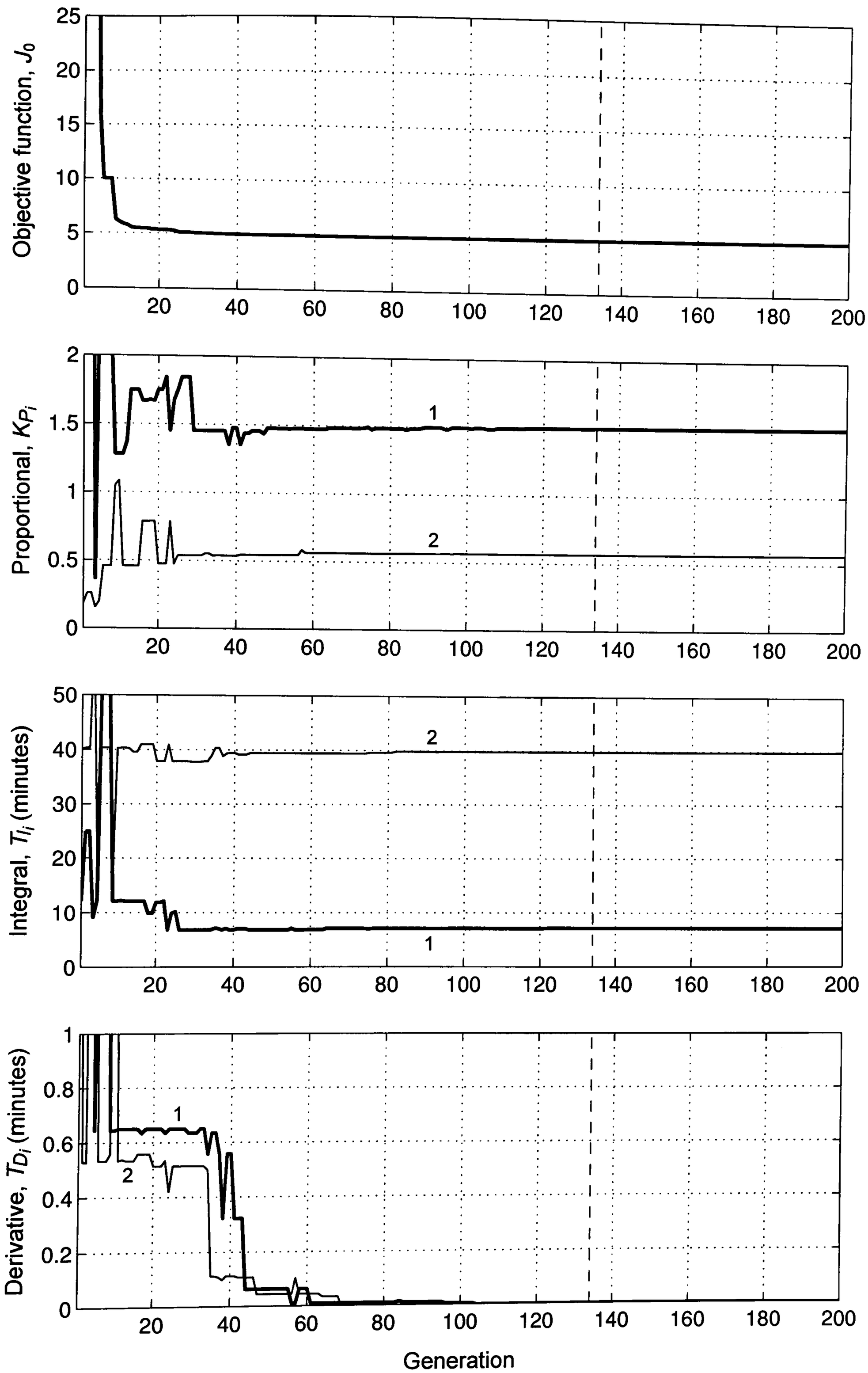


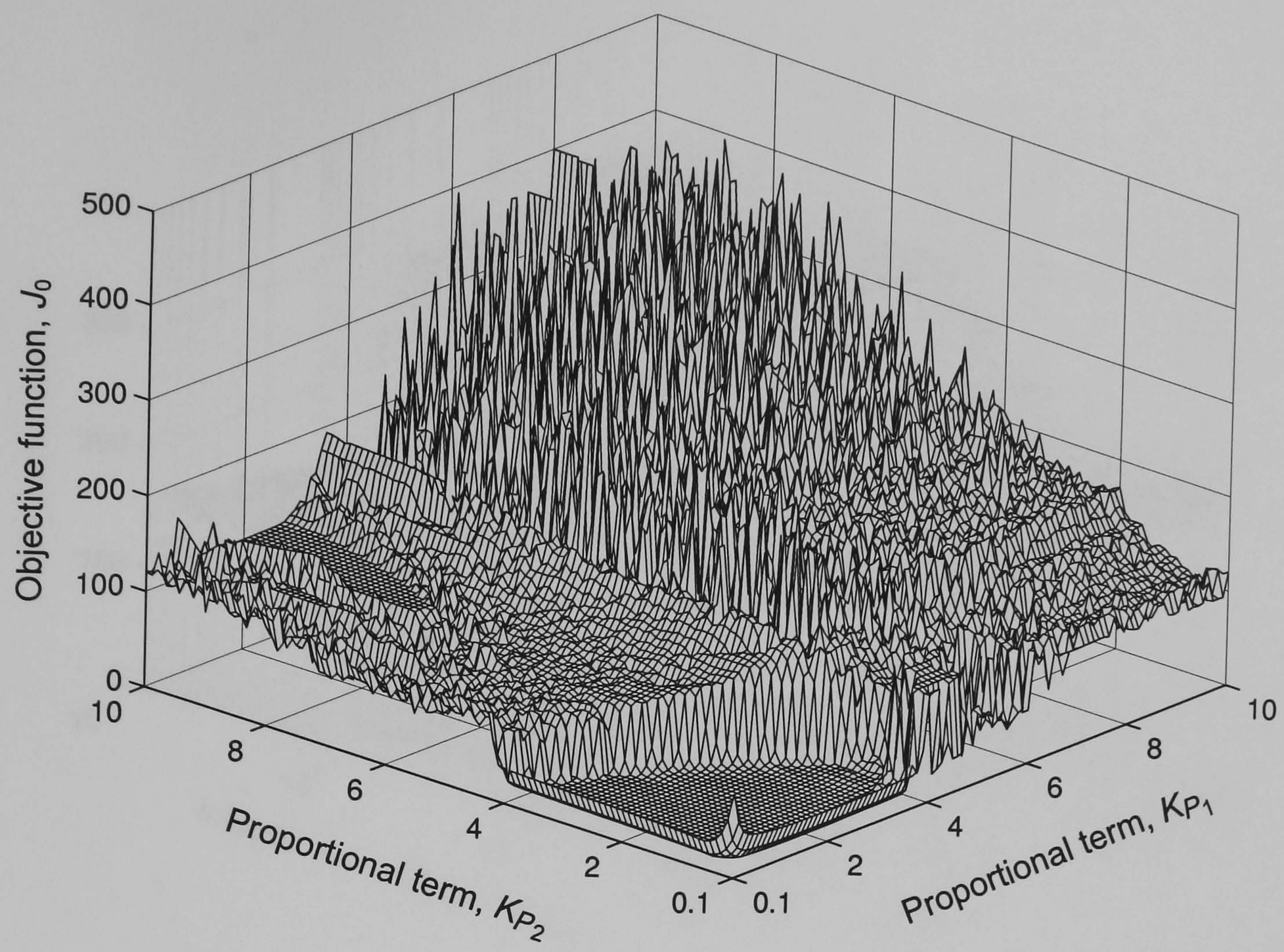
Fig. 6.24 *Convergence of J_0 and the six PID controller parameters*

In order to evaluate the performance of the GA in terms of optimality of the obtained solution, the minimisation of J_0 was repeated several times, resulting in a total of 50 independent GA runs. In each run, the GA was initialised with a different, randomly selected population. It was observed that in 72% of all runs (36 runs), the GA converged precisely to the solution shown in Table 6.2. In the remaining 28% (14 runs), the GA converged to solutions which, although sub-optimal, are practically identical to the one shown in Table 6.2, differing only in the terms K_{P_1} and T_{I_1}, where the observed absolute maximum differences were 7.3×10^{-3} (which is equivalent to three least significant bits or 0.073% of the search interval for K_{P_1}) and 2.4×10^{-2} minutes (which is equivalent to one least significant bit or 0.024% of the search interval for T_{I_1}), respectively. Of course, there is no guarantee that the solution shown in Table 6.2 is the one which achieves the global minimum of J_0, but the fact that practically all randomly initialised GA runs successfully converged to that particular solution is a strong indication of the optimality of the solution.

6.5.4 Search Landscape Complexity

The search landscape associated with the PID controller tuning is six-dimensional, and thus cannot be visualised easily. It is possible, however, to examine the surface slices that result by keeping any four of the six parameters at their optimal values shown in Table 6.2, and varying the remaining two. Gridding can then be used to compute the locations of the minima of the surface slice. In this way, the complexity of the search landscape can be assessed.

Three such surface slices and the corresponding maps of minima, obtained by varying the proportional, integral, and derivative terms, are shown in Figures 6.25, 6.26, and 6.27, respectively. The remaining controller parameters in each surface slice were kept at their optimal values shown in Table 6.2. It is observed that the surface slices are complex, highly non-linear and also multimodal. The multimodality of the surface slices can better be observed by examining the corresponding maps of minima, which were computed by evaluating J_0 at a grid of 501×501 points, where thousands of minima were found in each surface slice, of which only 1 (indicated by the circular marker) corresponds to the optimal controller parameters (unique solution).



Map of minima of surface slice – I and D terms at optimal values
Total number of minima: 21,184 – Optimal: 1 (unique solution)

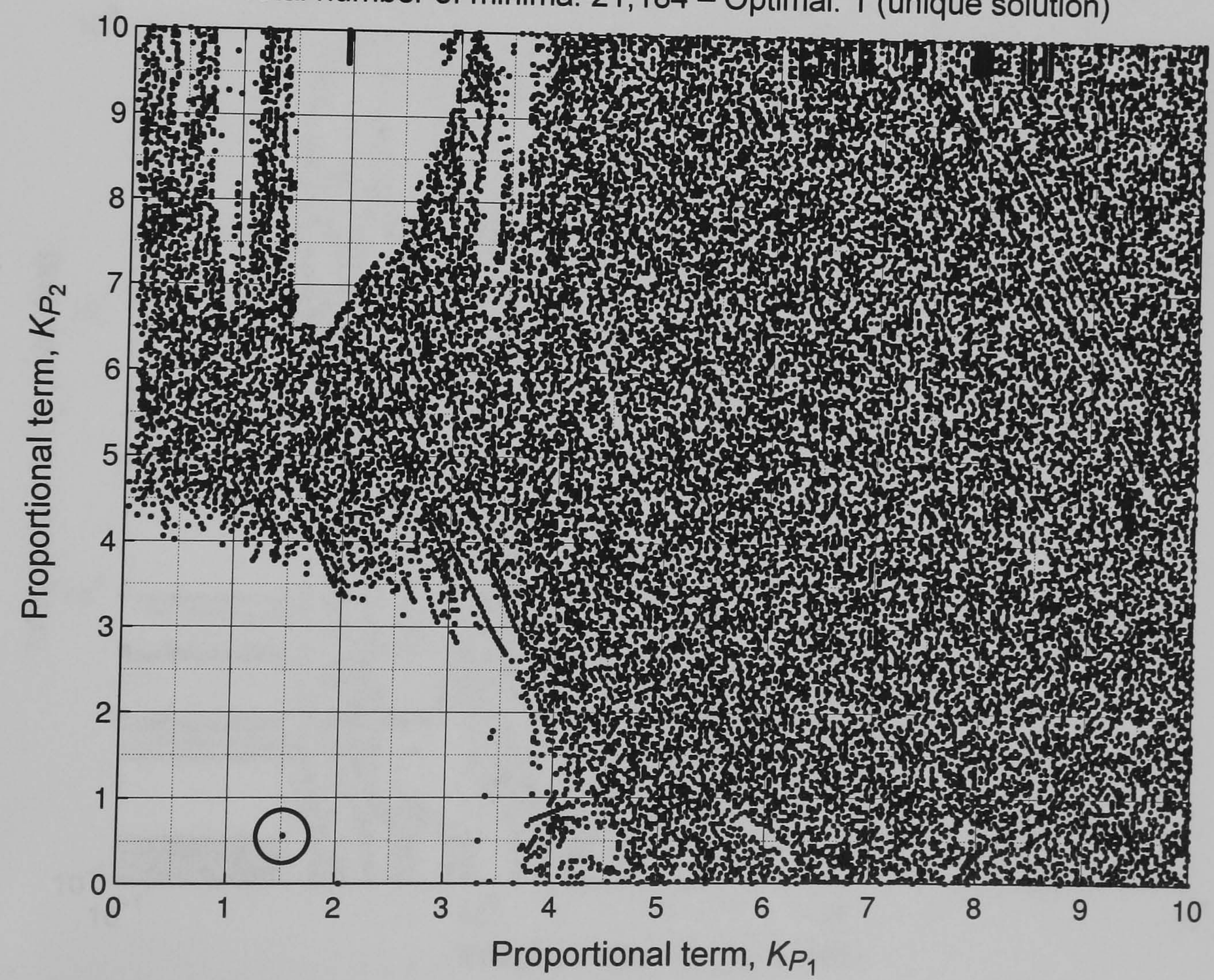
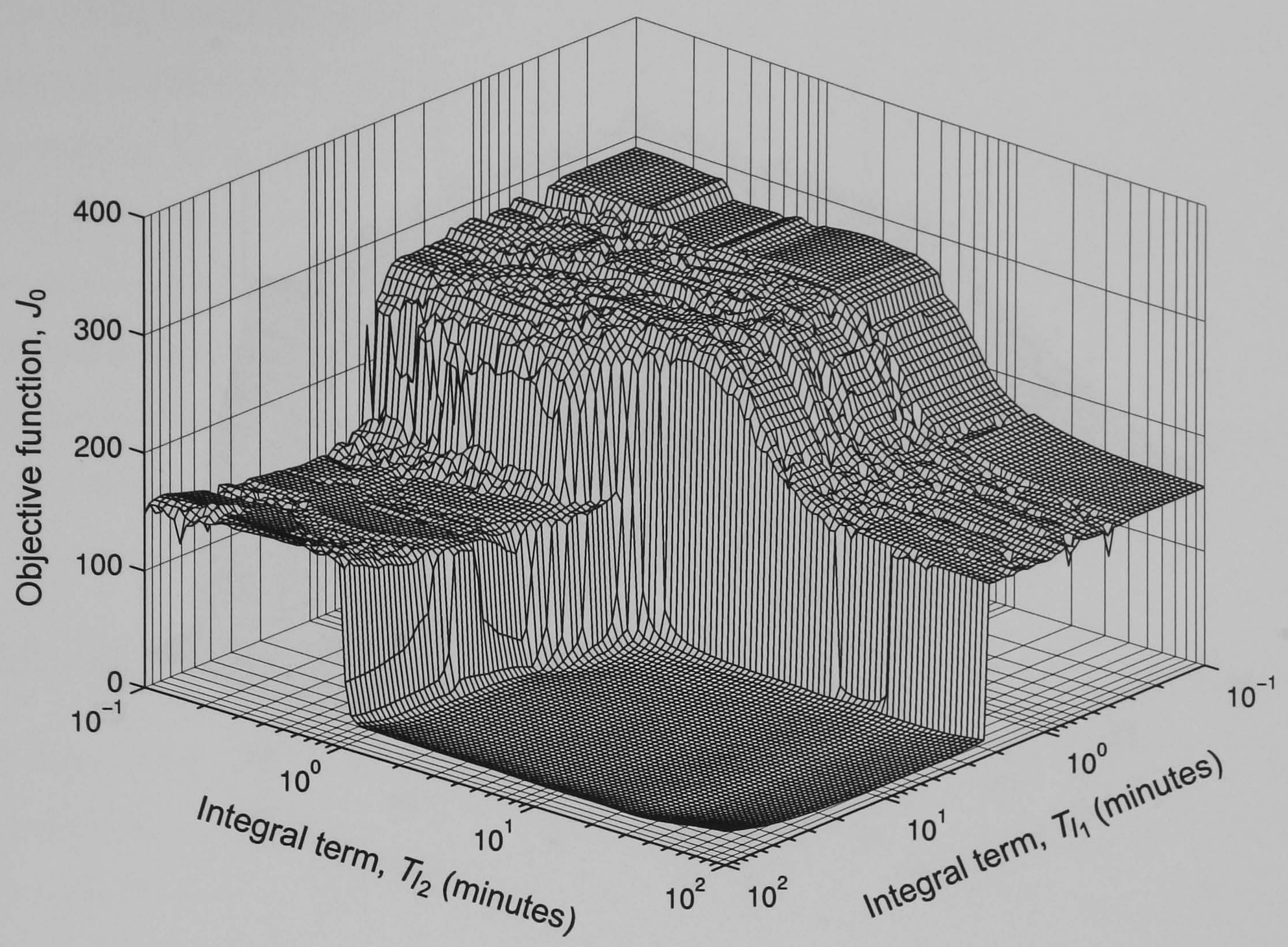


Fig. 6.25 *Objective function surface slice and map of minima (all integral and derivative terms are at their optimal values shown in Table 6.2)*



Map of minima of surface slice – P and D terms at optimal values
Total number of minima: 8,123 – Optimal: 1 (unique solution)

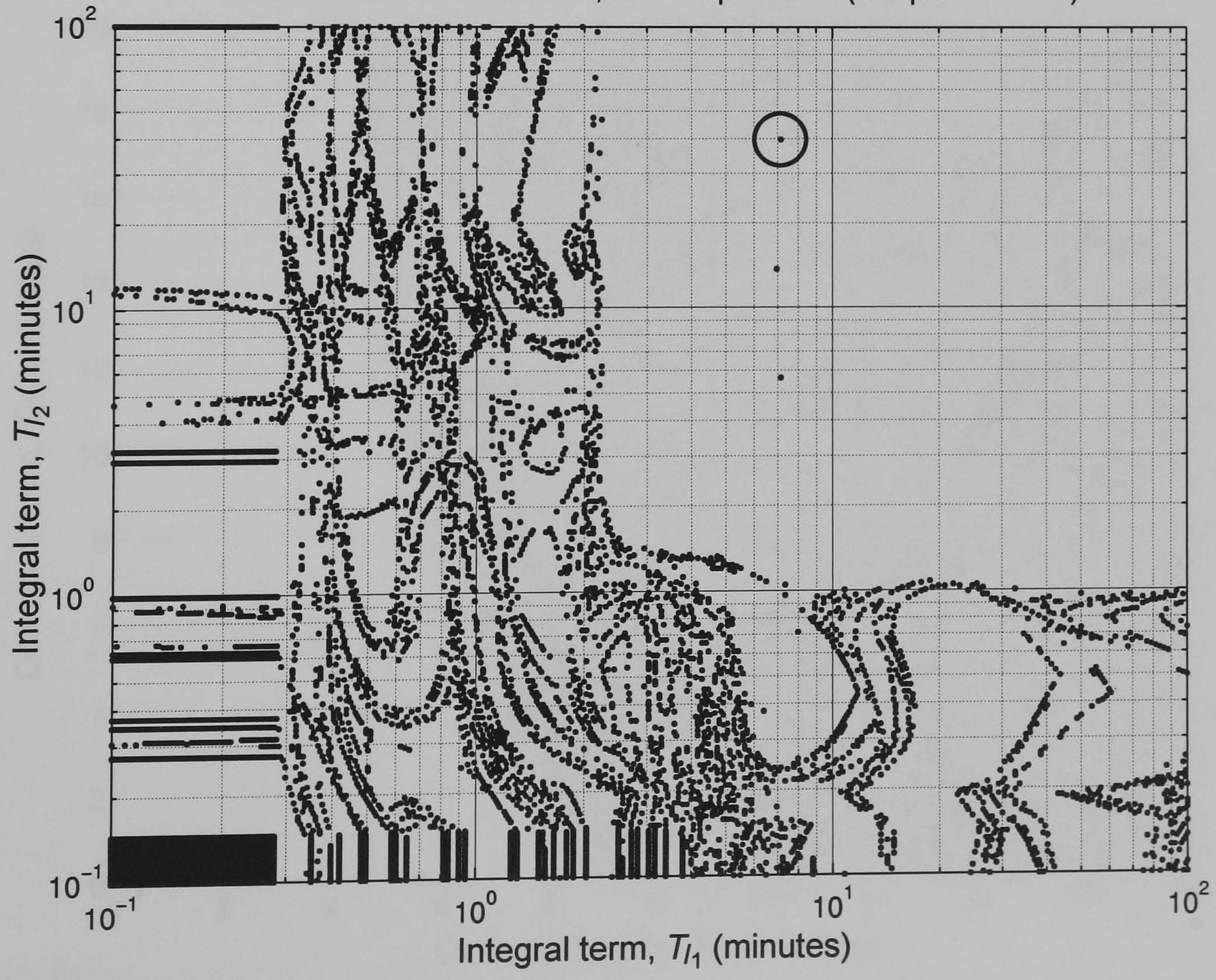
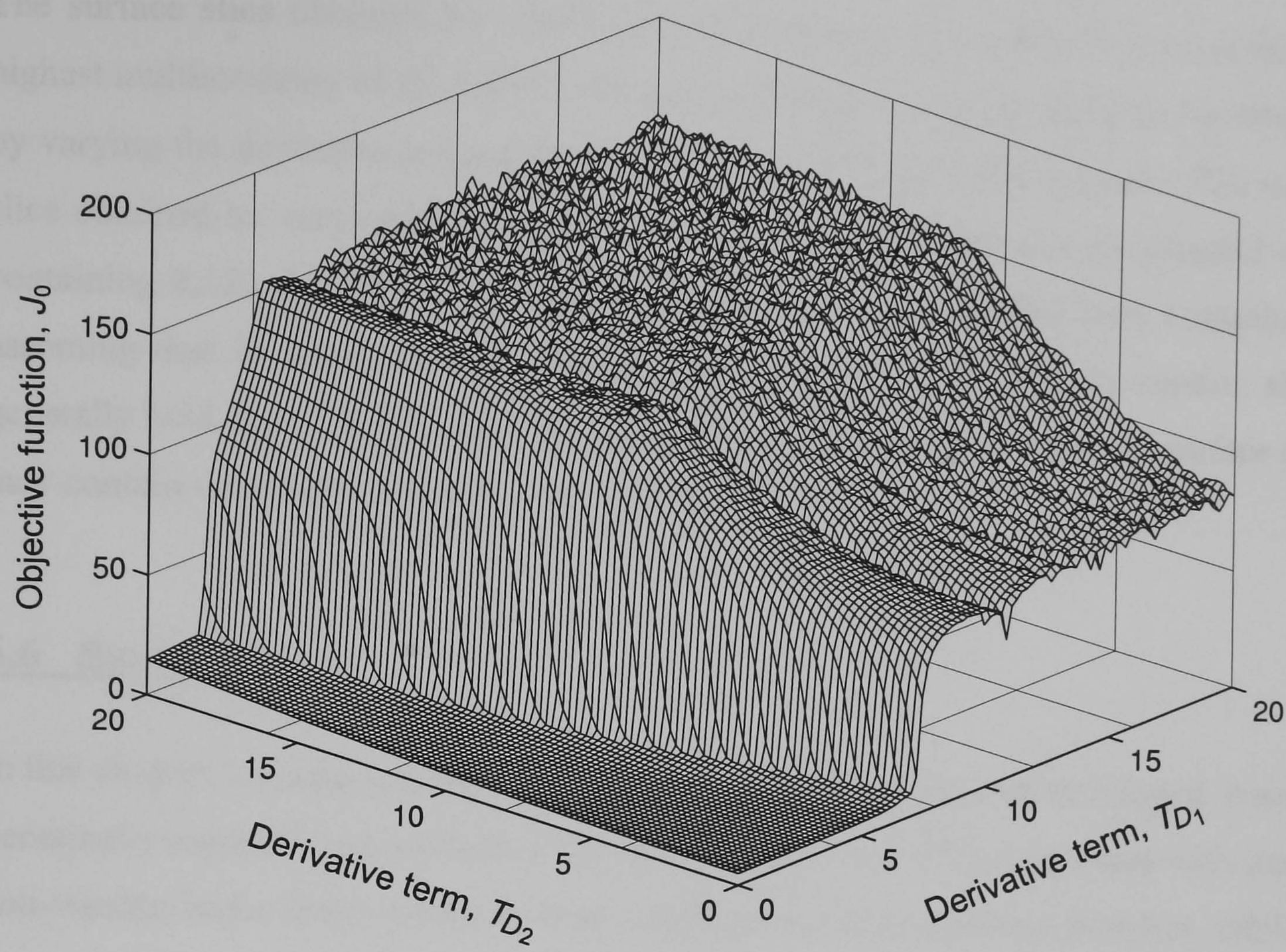


Fig. 6.26 *Objective function surface slice and map of minima (all proportional and derivative terms are at their optimal values shown in Table 6.2)*



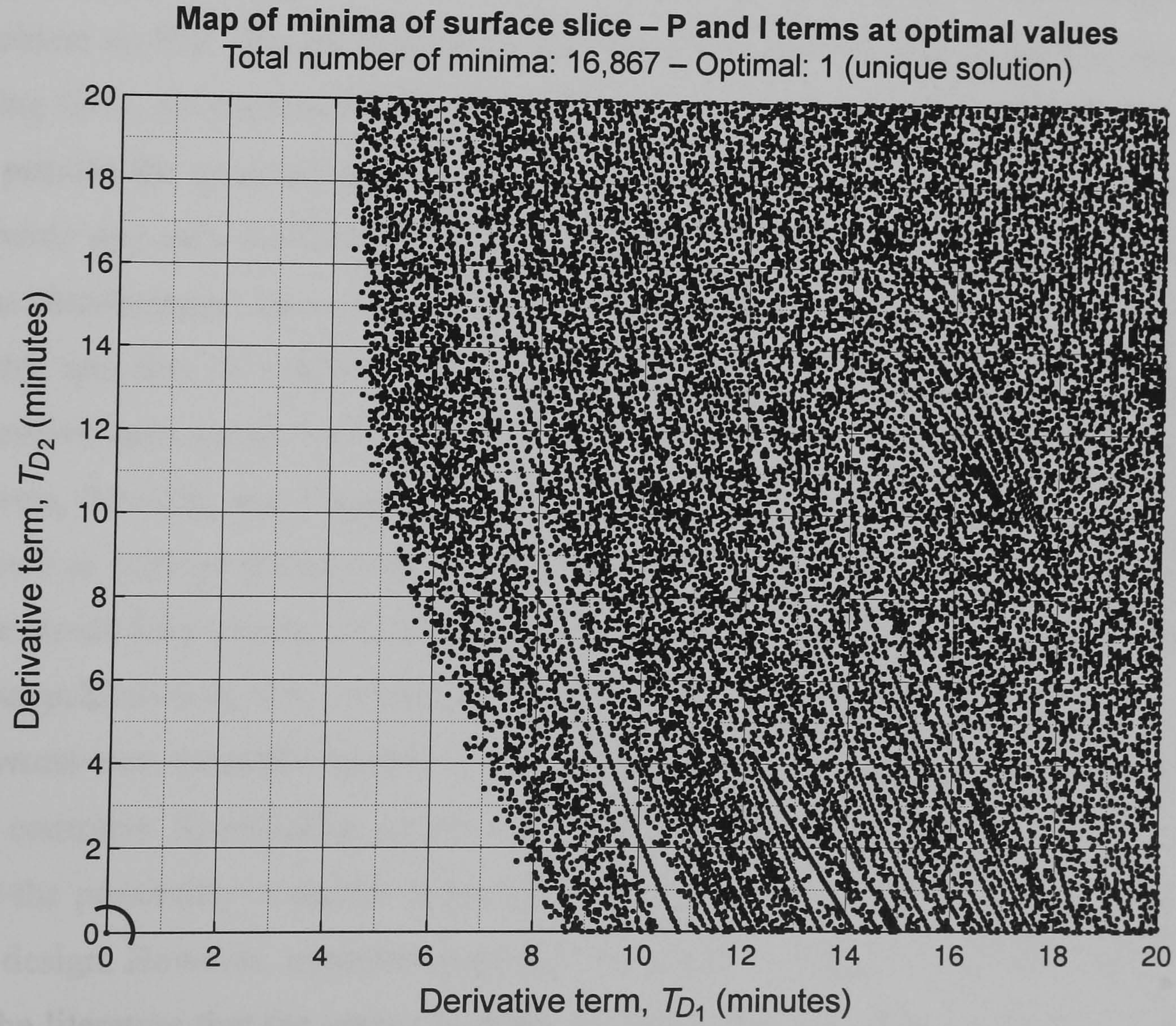


Fig. 6.27 *Objective function surface slice and map of minima (all proportional and integral terms are at their optimal values shown in Table 6.2)*

The surface slice obtained by varying the proportional terms (Fig. 6.25) exhibits the highest multimodality of all, with 21,184 minima, followed by the surface slice obtained by varying the derivative terms (Fig. 6.27), which contains 16,867 minima. The surface slice obtained by varying the integral terms (Fig. 6.26) was the least multimodal of all, containing 8,123 minima. Note that, in all three cases, the minima were computed by assuming that J_0 changes monotonically between grid points. This assumption should generally hold true if the grid is sufficiently dense. The actual 'continuous' surface slices may contain a much larger number of minima.

6.6 Summary

In this chapter, a solution to the Shell standard control problem was developed, based on genetically tuned PID controllers. Two linear, discrete-time PID controllers with integral anti-windup and a Smith predictor were employed for the regulation problem, while the input minimisation problem was solved analytically, by estimating the two unmeasured disturbances entering the process, and then solving the associated linear programming problem on-line. The six parameters associated with the two PID controllers were tuned using GAs. An extension of the objective function J_M developed in Chapter 3 was used to provide the necessary performance indexes. The performance of the proposed control scheme was very satisfactory, and this was illustrated by extensive simulation results. It was demonstrated, through simulation, that the resulting closed-loop system is robustly stable and that its robust performance is comparable to that of more computationally intensive approaches, such as QDMC and other algorithms (see, for example, Cuthrell, Rivera, Schmidt, and Vegeais, 1990). Making the specifications more conservative was shown to improve the robustness of the scheme in the face of large model uncertainties. The closed-loop system satisfied all steady-state specifications for all five prototype test cases published by Shell, which include the nominal process model as well as a number or worst-case uncertain models. It must be mentioned here that the proposed solution is not complete. Specifically, output constraint violations can arise in the transient period, and the possibility of failure in the sensors for outputs y_1 and y_2 was not considered in the design. However, transient constraint violations are present in all solutions available in the literature that the author is aware of, and it has been conjectured that a complete solution to the problem does not exist (Prett, García, and Morari, 1990). The optimality of the genetically tuned PID controllers was supported by repeating the controller tuning

several times, where it was observed that practically all randomly initialised GA runs successfully converged to the same set of controller parameters. The complexity of the search landscape associated with the PID tuning was demonstrated by examining three surface slices, where a large number of local minima were shown to be present. This justifies the use of GAs in this problem, as conventional optimisers are not likely to perform well in such complex search landscapes.

7 Conclusions – Main Contributions and Further Work

7.1 Introduction

The first part of this chapter summarises the key results and main contributions of this research project. A number of recommendations for further work in this direction, that will extend the application of GAs in the area of control systems engineering, are given in the second part of this chapter.

7.2 Summary of Main Contributions

This research work investigated the potential of the use of GAs as a basis for the optimal solution of control engineering problems in a function optimisation framework, focusing on multivariable process control. A number of novel performance indexes and controller design/tuning methods were developed and analysed, and an extension of the standard GA was proposed, which enables GAs to identify multiple equivalent optimal solutions to a given problem. The proposed methods were tested on a number of control problems involving multivariable processes of varying complexity, including a 5-input, 7-output chemical process with strong interactions, constraints, actuator non-linearities, and large dead times. The main contributions and novel aspects of this work can be summarised as follows. The references of the published parts of this work are also given.

7.2.1 Novel Objective Function Formulation for Control Systems

A novel objective function, denoted by $J_S(\cdot)$, was proposed for use in SISO systems, that enables the designer to *explicitly* specify the performance specifications associated with a given problem, in terms of time-domain bounds on the closed-loop system responses.

The formulation of J_S is such that the set of all controllers that completely satisfy the specifications is precisely the kernel of J_S. This set, denoted by $\tilde{\mathfrak{D}}$, can be the empty set (in cases where the given specifications are unachievable by all controllers in the search space), or can even be an *infinite* set, where an infinite number of equivalent (in terms of performance index) controllers exist that completely satisfy the given specifications. Objective function J_S was experimentally analysed using a simple PI controller tuning problem. The obtained solutions were compared with those obtained using conventional objective functions, as well as using several standard PI controller tuning methods. It was shown that J_S is capable of accurately quantifying complex performance specifications which cannot be accurately expressed in conventional terms such as gain/phase margin requirements. Finally, a more general objective function was proposed, denoted by $J_M(\cdot)$, which is a generalisation of J_S for use in MIMO systems. Parts of these results have been published in Vlachos, Evans, and Williams (1997), and Vlachos, Williams, and Gomm (1999a, 1999b).

7.2.2 Adaptive Fitness Sharing – An Extension of the Standard GA

A new method called *adaptive fitness sharing* was proposed, whose purpose is to enable GAs to locate multiple equivalent optimal solutions and distribute the members of the population uniformly within the optimal solution set. The proposed method is based on the techniques of *niche formation* and *speciation*, and is applicable to the optimisation of search landscapes which contain an infinite number of equivalent optimal solutions which share a unique objective function value that must be known *a priori*. The proposed method can thus be used for the optimisation of objective functions J_S and J_M. Formulae were derived for the estimation of the optimal value of the sharing radius σ_{share} involved in the fitness sharing algorithm. The optimality of σ_{share} is maintained during the course of the search run, by dynamically modifying the scaling of the parameter vectors. This was shown to be equivalent to the automatic adaptation of σ_{share}. The proposed method can thus be used in cases where the search landscape changes during the course of a GA search run. The time complexity of the proposed method was shown to be better than that of conventional fitness sharing. The computation time required to apply adaptive fitness sharing is usually much less than that of the objective function evaluations. Hence, the proposed method is not likely to significantly slow down the GA, and its application is simple and straightforward. The effectiveness of adaptive fitness sharing was supported

by extensive simulation results, and two population diversity measures were developed in order to quantify the obtained results. It was experimentally shown that adaptive fitness sharing outperforms the simple GA and population ranking alone. Furthermore, a set of statistical tests were performed, where it was clearly shown that adaptive fitness sharing consistently outperforms the simple GA and population ranking alone, in terms of both performance (higher degree of achieved uniformity) and robustness (less sensitivity to initial conditions). The adaptive properties of the proposed method were demonstrated by modifying the search landscape during the course of a GA run, where it was shown that adaptive fitness sharing successfully adapts the density of the population as required, while maintaining a high degree of uniformity throughout the search run.

7.2.3 Parametric Controller Tuning for Multivariable Processes

A new method for the automatic tuning of decentralised PI controllers for multivariable processes, based on GAs, was proposed. The major advantage of the proposed method is the ability to handle *arbitrary* performance specifications in the time-domain, that can be different for each system output. This is achieved by transforming the PI tuning problem into an optimisation problem, using objective function J_M. Adaptive fitness sharing is also employed, in order to maximise the diversity of the obtained family of optimal PI controllers. The numerical robustness and open architecture of GAs make the proposed method directly applicable to the automatic tuning of a wide range of linear or non-linear multivariable controllers, and not just PI controllers. The effectiveness of the proposed tuning method was supported by a number of simulation results using three two-input, two-output processes with different degrees of interaction between the two loops. It was shown that, in all cases, the resulting PI controllers completely satisfied all performance specifications. Adaptive fitness sharing was shown to achieve a high degree of diversity in the obtained family of optimal solutions. The choice of GAs as a suitable optimisation method was supported by comparing GAs with two conventional optimisation methods, where it is was shown that GAs have higher success rates and are more immune to noise. The complexity of the optimisation problem associated with one of the three PI controller tuning examples was experimentally demonstrated by examining two surface slices of the resulting four-dimensional search landscape, where a large number of local minima were shown to be present. Parts of these results have been published in Vlachos, Evans, and Williams (1997), and Vlachos, Williams, and Gomm (1999a, 1999b).

7.2.4 Solution to the Shell Standard Control Problem

A new solution to the Shell standard control problem was proposed, based on genetically tuned PID controllers. Two discrete-time PID controllers with integral anti-windup and a multivariable Smith predictor were employed for the output regulation problem, while the input minimisation problem was solved analytically, by estimating the unmeasured disturbances entering the process, and then solving the associated linear programming problem on-line. The six parameters associated with the two PID controllers were tuned using GAs. An extension of the objective function J_M was used to provide the necessary performance indexes. Extensive simulation results were presented, which show that the proposed control scheme achieves a very satisfactory performance. It was demonstrated, through simulation, that the resulting closed-loop system is robustly stable and that its robust performance is comparable to that of more computationally intensive approaches, such as the Quadratic Dynamic Matrix Control (QDMC) and other algorithms. Making the specifications more conservative was shown to improve the robustness of the scheme in the face of model uncertainties. It was also demonstrated, through simulation, that the proposed control scheme satisfies all steady-state specifications for all five prototype test cases published by Shell, which include the nominal process model as well as a number of worst-case uncertain models. The complexity of the optimisation problem associated with the PID controller tuning was demonstrated by examining three surface slices of the resulting six-dimensional search landscape, where a large number of local minima were shown to be present. This justifies the use of GAs in this problem, since conventional optimisers are not likely to perform well in search landscapes of such complexity. Parts of these results have been published in Vlachos, Williams, and Gomm (1998).

7.3 Recommendations for Further Work

In this section, a number of recommendations for further work are given, focusing on the development of improved objective functions for control systems, and on the application of GAs in multiobjective optimisation and real-time/adaptive control.

7.3.1 Improved Objective Functions for Control Systems

The non-dependence of GAs on continuity and derivative existence enables a wide range of complex objective functions for control systems to be developed, which accurately

quantify the performance specifications for a given control problem. Functions J_S and J_M, developed in this work, are only two such examples. Objective functions of such complexity would normally be avoided, due to the apparent lack of efficient optimisation methods. The robustness and global optimisation ability of GAs enable many complex objective functions to be developed, without the limitations imposed by continuity and derivative existence requirements. This approach can result in powerful methods which have the potential to deliver excellent designs that would normally be unachievable using conventional design methods based on linear, time-invariant control theory.

7.3.2 Multiobjective Optimisation

Most approaches to optimisation-based controller design and tuning use single-valued objective functions to provide the necessary performance indexes to guide the search for optimal solutions. However, control engineering problems are very seldom associated with a single objective. Instead, several, often conflicting objectives are usually present, thus resulting in vector-valued objective functions. Such cases are usually treated by weighting and combining all objectives into a single-valued function, thus transforming them into single-objective optimisation problems. Function J_M, developed in this work, uses this weighted-sum approach to combine all individual objectives J_{ij} by means of the weighting factors w_{ij}. This approach may be acceptable in certain cases, but there are times when combining the objectives in an efficient way may not be practically feasible. Although GAs are inherently unsuitable for multiobjective optimisation in their standard form, a number of extensions have been proposed (Fonseca and Fleming, 1995), which enable them to efficiently optimise vector-valued objective functions in a multiobjective framework. A vector-valued version of J_M can easily be obtained by simply redefining it as $\mathbf{J}_M := \left[\mathbf{J}_1 \cdots \mathbf{J}_q\right]^T \in \mathbb{R}_+^{q^2}$, where $\mathbf{J}_i = \left[J_{i1} \cdots J_{iq}\right] \in \mathbb{R}_+^q$. A multiobjective GA can then be employed to optimise $\mathbf{J}_M$. This approach eliminates the need to choose values for w_{ij}, and should greatly improve the efficiency of the resulting design/tuning methods.

7.3.3 Real-Time and Adaptive Control

The structure and operation of standard GAs generally prevent them from being applied to areas such as real-time and adaptive control, where the GA may be required to update certain system parameters on-line. The main problem is that, due to the stochastic nature

of their operation, standard GAs often produce solutions that cannot be directly applied to the real process for safety and other reasons. Furthermore, every member of the GA population has to be tested on the real process, and this can slow down the convergence of the GA if the process under control is relatively slow (as in most chemical processes, for example). The investigation into possible ways of extending standard GAs, in order to make them more suitable for real-time and adaptive control applications is a promising research area. In a process control framework, one possibility may be to develop a model of the process to be controlled, and use this model in a simulator to evaluate the members of the GA population at each generation. The solutions that result in stable closed-loops and achieve acceptable performance can then be selected and applied to the real process, and the objective function values of all members of the population can be corrected as necessary, based on the *true* performance of the selected solutions. In this way, only the well-behaved solutions are applied to the real process, thus minimising the probability of damaging the process, while at the same time the speed of convergence of the GA is greatly improved by only using the simulator to evaluate the solutions that appear to be unsatisfactory or inapplicable. Depending on the accuracy of the process model and the speed of response of the real process, a solution acceptability criterion can be specified that determines whether a candidate solution is satisfactory and safe enough to be tested on the real process. The acceptability criterion can be based on the candidate solution's objective function value, as well as on other additional safety criteria and the information acquired from previous on-line tests.

7.4 Summary

In this chapter, a summary was given of the key results and main contributions of this research project. A number of recommendations for further work in this direction, that will extend the application of GAs in the area of control systems engineering, were then outlined, concentrating on the development of improved objective functions for control systems, and on the use of GAs in multiobjective optimisation and real-time/adaptive control applications.

A The Shell Standard Control Problem (Prett and Morari, 1987)

A.1 Problem Description

Fig. A.1 shows a heavy oil fractionator with three product draws and three side circulating loops. The heat requirement of the column enters with the feed, which is a gaseous stream. Product specifications for the top and side draws are determined by economics and operating requirements. There is no product specification for the bottom draw, but there is an operating constraint on the temperature in the lower part of the column. The three circulating loops remove heat to achieve the desired product separation. The heat exchangers in these loops reboil columns in other parts of the plant. Therefore, they have varying heat duty requirements. The bottom loop has an enthalpy controller which regulates heat removal in the loop by adjusting steam make. Its heat duty can be used as a manipulated variable to control the column.

The relevant information regarding the Shell standard control problem is stated in the following five sections.

1. Control objectives
2. Control constraints
3. Process model
4. Uncertainties in the gains of the model
5. Prototype test cases

We have tried to encapsulate the relevant control issues in this one problem while staying as realistic as possible. The problem is stated such that an infinite number of scenarios can occur in controlling the unit. We would encourage the development of solution methodologies that are flexible enough to deal with varying (and possibly conflicting) problem requirements, and can be readily automated such that control designs can be carried out by plant personnel with only a modest knowledge of control concepts.

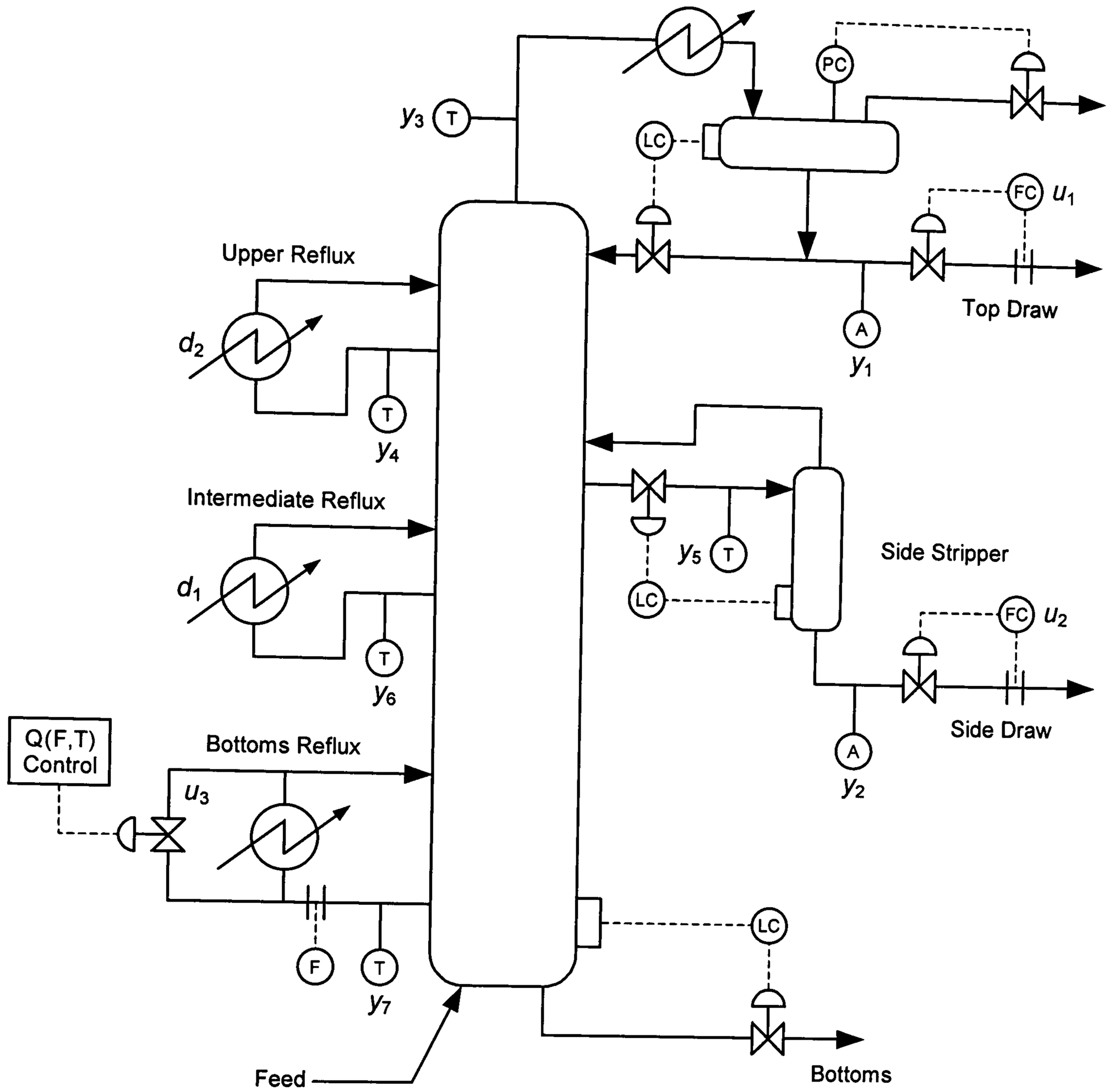


Fig. A.1 *Diagram of a heavy oil fractionator (Shell control problem)*

A complete solution to the problem should describe, in detail, the analysis and synthesis procedures that indicate that the proposed controller satisfies the control objectives for all plants in the uncertainty set. However, because of possible discrepancies between investigators on analysis techniques, we have formulated a number of prototype test cases which form a common frame of reference for evaluating different designs.

A.2 Control Objectives

1. Maintain the top and side draw product end points at specification (0.0 ±0.005 in the steady state).

2. Maximise steam make in the steam generators (maximise heat removal) in the bottom circulating reflux (Important note: Heat duties are expressed in terms of heat input to the column. Decreasing heat duty implies increasing the amount of heat removed).

3. Reject the unmeasured disturbances entering the column from the upper and intermediate refluxes due to changes in heat duty requirements from other columns (upper and intermediate reflux duties range between –0.5 and 0.5). Reject disturbances even when one or both end point analysers fail.

4. Keep the closed-loop speed of response between 0.8 and 1.25 of the open-loop process bandwidth.

A.3 Control Constraints

1. All draws must be within hard maximum and minimum bounds of 0.5 and –0.5.
2. The bottom reflux heat duty is constrained within the hard bounds of 0.5 and –0.5.
3. All manipulated variables have maximum move size limitations of magnitude 0.05 units per minute.
4. Fastest sampling time is 1 minute.
5. The bottom reflux draw temperature has a minimum value of –0.5.
6. The top end point must be maintained within the maximum and minimum values of 0.5 and –0.5.

A.4 Process Model (First-Order Dead Time)

The model of the Shell heavy oil fractionator process is a transfer function matrix $\mathbf{G}(s)$, whose i, j element is a first-order dead time transfer function that relates the i-th process output with the j-th process input.

$$\mathbf{G}(s) = \begin{bmatrix} G_{11}(s) & G_{12}(s) & \cdots & G_{15}(s) \\ G_{21}(s) & G_{22}(s) & \cdots & G_{25}(s) \\ \vdots & \vdots & \ddots & \vdots \\ G_{71}(s) & G_{72}(s) & \cdots & G_{75}(s) \end{bmatrix} \tag{A.1}$$

$$G_{ij}(s) = \frac{K_{ij} e^{-L_{ij} s}}{T_{ij} s + 1} \tag{A.2}$$

The nominal model gains K_{ij}, time constants T_{ij}, and time delays L_{ij}, are all shown in Table A.1. The units for T_{ij} and L_{ij} are in minutes.

TABLE A.1 Shell heavy oil fractionator model nominal parameters, K_{ij}, T_{ij}, and L_{ij}

	Top draw (u_1)			Side draw (u_2)			Bottoms reflux duty (u_3)			Inter. reflux duty (d_1)			Upper reflux duty (d_2)		
	K_{ij}	T_{ij}	L_{ij}	K_{ij}	T_{ij}	L_{ij}	K_{ij}	T_{ij}	L_{ij}	K_{ij}	T_{ij}	L_{ij}	K_{ij}	T_{ij}	L_{ij}
Top end point (y_1)	4.05	50	27	1.77	60	28	5.88	50	27	1.20	45	27	1.44	40	27
Side end point (y_2)	5.39	50	18	5.72	60	14	6.90	40	15	1.52	25	15	1.83	20	15
Top temperature (y_3)	3.66	9	2	1.65	30	20	5.53	40	2	1.16	11	0	1.27	6	0
Upper reflux temp. (y_4)	5.92	12	11	2.54	27	12	8.10	20	2	1.73	5	0	1.79	19	0
Side draw temp. (y_5)	4.13	8	5	2.38	19	7	6.23	10	2	1.31	2	0	1.26	22	0
Inter. reflux temp. (y_6)	4.06	13	8	4.18	33	4	6.53	9	1	1.19	19	0	1.17	24	0
Bottoms reflux temp. (y_7)	4.38	33	20	4.42	44	22	7.20	19	0	1.14	27	0	1.26	32	0

A.5 Uncertainties in the Gains of the Model

The gains of the transfer function elements of the process model $\mathbf{G}(s)$ are subject to uncertainties of the following form.

$$G_{ij}(s) = \frac{K_{ij} + \varepsilon_j \Delta K_{ij}}{T_{ij} s + 1} e^{-L_{ij} s}, \quad -1 \le \varepsilon_j \le 1 \tag{A.3}$$

where ΔK_{ij} denotes the absolute maximum uncertainty of gain K_{ij}, and ε_j determines the amount of uncertainty of gain K_{ij}. The values of ΔK_{ij} are shown in Table A.2.

TABLE A.2 Shell heavy oil fractionator model gain uncertainty parameters, ΔK_{ij}

	Top draw (u_1)	Side draw (u_2)	Bottoms reflux duty (u_3)	Inter. reflux duty (d_1)	Upper reflux duty (d_2)
Top end point (y_1)	2.11	0.39	0.59	0.12	0.16
Side end point (y_2)	3.29	0.57	0.89	0.13	0.13
Top temperature (y_3)	2.29	0.35	0.67	0.08	0.08
Upper reflux temp. (y_4)	2.34	0.24	0.32	0.02	0.04
Side draw temp. (y_5)	1.71	0.93	0.30	0.03	0.02
Inter. reflux temp. (y_6)	2.39	0.35	0.72	0.08	0.01
Bottoms reflux temp. (y_7)	3.11	0.73	1.33	0.18	0.18

A.6 Prototype Test Cases

Demonstrate, through simulation, that the proposed controller satisfies the control objectives without violating the control constraints for the following plants within the uncertainty set (assume all inputs and outputs are initially at zero). The magnitudes for the upper and intermediate reflux duty step changes are indicated below.

1. $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = \varepsilon_4 = \varepsilon_5 = 0$. Upper reflux duty = 0.5, intermediate = 0.5.
2. $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = -1$, $\varepsilon_4 = \varepsilon_5 = 1$. Upper reflux duty = –0.5, intermediate = –0.5.
3. $\varepsilon_1 = \varepsilon_3 = \varepsilon_4 = \varepsilon_5 = 1$, $\varepsilon_2 = -1$. Upper reflux duty = –0.5, intermediate = –0.5.
4. $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = \varepsilon_4 = \varepsilon_5 = 1$. Upper reflux duty = 0.5, intermediate = –0.5.
5. $\varepsilon_1 = -1$, $\varepsilon_2 = 1$, $\varepsilon_3 = \varepsilon_4 = \varepsilon_5 = 0$. Upper reflux duty = –0.5, intermediate = –0.5.

B Minimal State-Space Realisations of $\mathbf{G}_M(z)$, $\mathbf{G}_D^{-1}(z)$, $\mathbf{G}_S(z)$, and $\mathbf{G}_S^*(z)$

B.1 State-Space Model Representation

The discrete-time process models $\mathbf{G}_M(z)$, $\mathbf{G}_D^{-1}(z)$, $\mathbf{G}_S(z)$, and $\mathbf{G}_S^*(z)$, that were employed in the proposed GA-based solution to the Shell standard control problem (Chapter 6), are given in this Appendix. The models were derived from the nominal process model $\mathbf{G}(s)$ using the modified z-Transform method, and are expressed in the following standard discrete-time state-space representation.

$$\mathbf{x}[(k+1)T] = \mathbf{A}\mathbf{x}(kT) + \mathbf{B}\mathbf{u}(kT) \tag{B.1}$$

$$\mathbf{y}(kT) = \mathbf{C}\mathbf{x}(kT) + \mathbf{D}\mathbf{u}(kT) \tag{B.2}$$

The sample time for all four models is $T=5$ minutes. The **A**, **B**, **C**, and **D** matrices of the minimal state-space realisations of models $\mathbf{G}_M(z)$, $\mathbf{G}_D^{-1}(z)$, $\mathbf{G}_S(z)$, and $\mathbf{G}_S^*(z)$ are given in the following sections.

B.2 Disturbance Estimator Models

Models $\mathbf{G}_M(z)$, $\mathbf{G}_D^{-1}(z)$ are used in the disturbance estimator that is shown in Eq. (6.15), to estimate the unmeasured disturbances entering the Shell process. Model $\mathbf{G}_M(z)$ maps process inputs u_1, u_2, and u_3 to process outputs y_3 and y_5, while the inverse model $\mathbf{G}_D^{-1}(z)$ maps process outputs y_3 and y_5 to process disturbances d_1 and d_2. Recall that $\mathbf{G}_D^{-1}(z)$ can be expressed as

$$\mathbf{G}_D^{-1}(z) = \begin{bmatrix} G_{34}^*(z) & G_{35}^*(z) \\ G_{54}^*(z) & G_{55}^*(z) \end{bmatrix}^{-1} z^1 \tag{B.3}$$

The system shown in Eq. (B.3) is non-causal and is, therefore, unrealisable. The model used in the proposed solution is obtained by introducing a delay of one sample to $\mathbf{G}_D^{-1}(z)$, resulting in the realisable system $z^{-1}\mathbf{G}_D^{-1}(z)$.

The **A**, **B**, **C**, and **D** matrices of the minimal state-space realisation of $\mathbf{G}_M(z)$ are given below, with their elements truncated to five significant digits.

$$\mathbf{A} = \begin{bmatrix}
8.1123\times10^{-1} & 3.3034\times10^{-1} & 4.2353\times10^{-1} & -4.2429\times10^{-2} & 1.8497\times10^{-3} & -1.6913\times10^{-1} & -3.8037\times10^{-2} & -3.4886\times10^{-1} & 3.3621\times10^{-2} & -1.3253\times10^{-1} & 3.5936\times10^{-2} \\
2.6932\times10^{-2} & 5.6513\times10^{-1} & -5.6254\times10^{-2} & -1.0648\times10^{-1} & 1.2335\times10^{-1} & -5.7893\times10^{-2} & -1.8487\times10^{-1} & 4.7635\times10^{-1} & -9.4686\times10^{-2} & 9.0974\times10^{-3} & 1.7006\times10^{-2} \\
1.6144\times10^{-2} & -6.5287\times10^{-2} & 6.6293\times10^{-1} & -1.3177\times10^{-1} & -2.4529\times10^{-1} & -3.0730\times10^{-1} & 5.5904\times10^{-2} & 5.2455\times10^{-1} & 1.6653\times10^{-1} & 1.1766\times10^{-1} & 3.7046\times10^{-2} \\
0 & -1.7494\times10^{-3} & 2.0130\times10^{-1} & 5.6862\times10^{-1} & -3.7132\times10^{-1} & -1.2348\times10^{-1} & 2.7045\times10^{-1} & 1.7523\times10^{-1} & -1.3908\times10^{-1} & -7.9846\times10^{-2} & -4.8762\times10^{-2} \\
0 & 1.2757\times10^{-1} & 2.7603\times10^{-3} & 2.7728\times10^{-1} & 8.3980\times10^{-2} & 1.0125 & -8.8723\times10^{-2} & -7.0760\times10^{-3} & -2.5415\times10^{-2} & 2.4044\times10^{-3} & -3.8019\times10^{-2} \\
0 & 0 & 0 & -1.0705\times10^{-1} & 2.8927\times10^{-1} & 2.5506\times10^{-1} & 1.0342\times10^{-1} & -2.1710\times10^{-2} & 1.7797\times10^{-2} & -4.2364\times10^{-2} & 5.2773\times10^{-2} \\
0 & 0 & 0 & 4.2573\times10^{-2} & 7.2735\times10^{-1} & -2.2925\times10^{-1} & 1.8366\times10^{-1} & 1.1180\times10^{-1} & 2.2709\times10^{-2} & -2.1036\times10^{-2} & 6.9740\times10^{-2} \\
0 & 0 & 0 & 0 & 0 & -2.3133\times10^{-2} & 8.8901\times10^{-1} & -5.8348\times10^{-2} & 4.0263\times10^{-2} & -1.5963\times10^{-1} & 1.8985\times10^{-2} \\
0 & 0 & 0 & 0 & 0 & -1.3037\times10^{-1} & -1.5774\times10^{-1} & 1.2859\times10^{-1} & -1.1155\times10^{-1} & -2.4172\times10^{-1} & -9.5897\times10^{-2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.8349\times10^{-1} & 4.8268\times10^{-1} & 7.6366\times10^{-1} & 4.4344\times10^{-2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.4125\times10^{-1} & -6.2704\times10^{-1} & -2.4721\times10^{-1} & 4.8877\times10^{-1}
\end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix}
-2.5957\times10^{-2} & 2.7364\times10^{-1} & -1.5407\times10^{-1} \\
-4.7853\times10^{-2} & -4.7602\times10^{-3} & 1.6352\times10^{-3} \\
4.6167\times10^{-1} & 6.0542\times10^{-1} & -6.8283\times10^{-1} \\
-2.2493\times10^{-1} & -1.3125\times10^{-1} & 1.3328\times10^{-1} \\
3.5091\times10^{-1} & -3.0657\times10^{-1} & -4.6210\times10^{-1} \\
3.7313\times10^{-1} & -1.1179 & -4.0604\times10^{-1} \\
1.2686\times10^{-1} & 2.7392\times10^{-1} & -4.6085\times10^{-1} \\
2.1675\times10^{-1} & 3.1699\times10^{-1} & -2.3821\times10^{-1} \\
8.0265\times10^{-1} & 1.4841\times10^{-1} & 5.3374\times10^{-1} \\
8.7365\times10^{-1} & 0 & 3.9904\times10^{-1} \\
1.8928\times10^{-2} & 0 & -6.3294\times10^{-1}
\end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.1850 & 1.1580\times10^{-1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5.4526\times10^{-2} & -2.5168
\end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Similarly, the **A**, **B**, **C**, and **D** matrices of the minimal state-space realisation of $z^{-1}\mathbf{G}_D^{-1}(z)$ are given below.

$$\mathbf{A} = \begin{bmatrix} 5.4102\times10^{-1} & 5.2205\times10^{-2} & -3.2764\times10^{-1} & 6.9073\times10^{-1} \\ 5.2205\times10^{-2} & 2.2599\times10^{-1} & 4.8360\times10^{-1} & -3.0540\times10^{-1} \\ -4.4631\times10^{-2} & 2.7653\times10^{-1} & 6.8188\times10^{-1} & -5.2987\times10^{-1} \\ 9.7708\times10^{-2} & 1.5770\times10^{-2} & -4.4385\times10^{-2} & 1.1407\times10^{-1} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} -1.4395 & 1.2484 \\ 7.7960\times10^{-1} & -9.1342\times10^{-1} \\ 1.2862 & -9.1117\times10^{-2} \\ 9.8509\times10^{-1} & 1.9579\times10^{-1} \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & -2.1563\times10^{-1} & 1.3979\times10^{-1} \\ 0 & 0 & 1.2724\times10^{-1} & -6.5620\times10^{-1} \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} -3.3932\times10^{-1} & 9.5119\times10^{-1} \\ 1.5929 & -5.6127\times10^{-1} \end{bmatrix}$$

B.3 Smith Predictor Models

Models $\mathbf{G}_S(z)$ and $\mathbf{G}_S^*(z)$ are used in the multivariable Smith predictor shown in Fig. 6.7. Model $\mathbf{G}_S(z)$ maps process inputs u_1, u_2, and u_3 to process outputs y_1 and y_2, while model $\mathbf{G}_S^*(z)$ is obtained by removing all delay states from all of the elements of $\mathbf{G}_S(z)$. The **A**, **B**, **C**, and **D** matrices of the minimal state-space realisation of $\mathbf{G}_S(z)$ are given below.

$$\mathbf{A} = \begin{bmatrix}
8.8756\times10^{-1} & -4.2295\times10^{-1} & 3.9721\times10^{-1} & -4.5609\times10^{-2} & -4.4462\times10^{-1} & -1.7091\times10^{-1} & -7.3164\times10^{-2} & 1.4749\times10^{-1} & 9.1540\times10^{-2} & -2.7887\times10^{-2} & 4.7660\times10^{-2} & -2.6611\times10^{-2} & -8.6266\times10^{-2} & 5.9327\times10^{-2} \\
-3.8448\times10^{-2} & 2.4002\times10^{-1} & 6.4848\times10^{-1} & -2.7843\times10^{-1} & 6.0858\times10^{-1} & 1.9770\times10^{-1} & 2.2259\times10^{-1} & -1.0443\times10^{-1} & -2.0312\times10^{-1} & 6.7762\times10^{-3} & 3.0895\times10^{-2} & 5.0364\times10^{-2} & -9.3126\times10^{-2} & -3.0116\times10^{-2} \\
-3.6169\times10^{-3} & 2.4074\times10^{-1} & 6.0398\times10^{-1} & 9.2950\times10^{-1} & -3.8137\times10^{-1} & 3.2563\times10^{-1} & 6.6621\times10^{-2} & 5.3136\times10^{-3} & -1.2692\times10^{-1} & 5.2196\times10^{-2} & -8.3305\times10^{-2} & 4.4659\times10^{-2} & 1.5800\times10^{-1} & -7.8008\times10^{-2} \\
-3.7844\times10^{-3} & -2.3009\times10^{-1} & 2.8134\times10^{-1} & 8.6864\times10^{-2} & 3.3954\times10^{-2} & -1.4049\times10^{-1} & -8.3515\times10^{-2} & 1.6038\times10^{-1} & 6.9712\times10^{-2} & 4.3052\times10^{-2} & 7.2564\times10^{-2} & -3.6700\times10^{-2} & -1.3811\times10^{-1} & 6.8522\times10^{-2} \\
0 & 0 & -2.1292\times10^{-2} & 2.9073\times10^{-1} & 4.0712\times10^{-1} & -1.3261\times10^{-1} & -1.0203\times10^{-1} & 1.2818\times10^{-1} & 8.5989\times10^{-2} & 4.1186\times10^{-3} & 2.7702\times10^{-4} & -2.2440\times10^{-2} & 1.3048\times10^{-2} & 2.7893\times10^{-2} \\
0 & 0 & 1.0603\times10^{-2} & 5.8383\times10^{-1} & 6.9825\times10^{-1} & -2.3335\times10^{-1} & -1.4592\times10^{-1} & -1.1177\times10^{-1} & 1.7268\times10^{-1} & 1.6344\times10^{-2} & 6.7231\times10^{-3} & -4.7653\times10^{-2} & 1.4040\times10^{-2} & 6.0780\times10^{-2} \\
0 & 0 & 0 & 0 & -1.5502\times10^{-2} & 7.5760\times10^{-1} & -2.4776\times10^{-1} & -8.0872\times10^{-2} & 3.4275\times10^{-1} & -3.4767\times10^{-3} & 1.3136\times10^{-1} & -1.0974\times10^{-1} & -2.1832\times10^{-1} & 1.9819\times10^{-1} \\
0 & 0 & 0 & 0 & 3.6320\times10^{-1} & 3.2335\times10^{-2} & -1.4312\times10^{-1} & 7.4975\times10^{-1} & 5.0173\times10^{-2} & -3.5549\times10^{-2} & 1.8547\times10^{-2} & -9.9299\times10^{-3} & -3.3131\times10^{-2} & 2.5246\times10^{-2} \\
0 & 0 & 0 & 0 & 0 & 0 & 6.8496\times10^{-1} & 5.3892\times10^{-1} & 3.9088\times10^{-1} & 2.8051\times10^{-2} & -4.8921\times10^{-2} & -8.0855\times10^{-2} & 1.8054\times10^{-1} & 1.7358\times10^{-1} \\
0 & 0 & 0 & 0 & 0 & 0 & 4.2294\times10^{-1} & -8.7278\times10^{-1} & 2.5807\times10^{-1} & 1.7996\times10^{-2} & 6.3371\times10^{-3} & -6.5061\times10^{-2} & 3.5851\times10^{-2} & 1.2203\times10^{-1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6.6479\times10^{-1} & 1.6959\times10^{-1} & 3.0435\times10^{-2} & 2.5691\times10^{-1} & -2.4848\times10^{-1} & -4.0389\times10^{-1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.1402\times10^{-1} & 9.8882\times10^{-1} & 2.0075\times10^{-2} & -2.9853\times10^{-2} & -2.3346\times10^{-2} & 5.4388\times10^{-2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8.1600\times10^{-1} & 1.0184\times10^{-1} & 4.5293\times10^{-1} & -9.2591\times10^{-2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -8.8361\times10^{-2} & 9.4046\times10^{-1} & 0 & 2.5567\times10^{-1}
\end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} -6.0704\times10^{-1} & 4.3012\times10^{-1} & 2.3708\times10^{-1} \\ 2.9394\times10^{-1} & -2.0828\times10^{-1} & -1.6119\times10^{-1} \\ -2.4076\times10^{-1} & -6.9959\times10^{-1} & 5.1415\times10^{-1} \\ 1.4635\times10^{-2} & -9.0709\times10^{-1} & 5.6393\times10^{-1} \\ 1.1784 & -3.4238\times10^{-2} & -3.9278\times10^{-2} \\ -9.0215\times10^{-2} & -2.2938\times10^{-1} & -7.7114\times10^{-2} \\ -4.9364\times10^{-2} & -9.7559\times10^{-2} & -1.8937\times10^{-1} \\ 2.9631\times10^{-1} & 6.1421\times10^{-1} & 1.1367 \\ 0 & 5.6308\times10^{-2} & 0 \\ 0 & -1.2503\times10^{-1} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5.0273\times10^{-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -7.6405\times10^{-1} \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Finally, the **A**, **B**, **C**, and **D** matrices of the minimal state-space realisation of $\mathbf{G}_S^*(z)$ are given below.

$$\mathbf{A} = \begin{bmatrix} 9.0253\times10^{-1} & -4.1354\times10^{-3} & -2.4712\times10^{-3} & 5.5050\times10^{-3} \\ -4.1354\times10^{-3} & 9.1539\times10^{-1} & -1.9989\times10^{-3} & 1.1153\times10^{-2} \\ -2.4712\times10^{-3} & -1.9989\times10^{-3} & 9.0252\times10^{-1} & 6.0768\times10^{-3} \\ 5.5050\times10^{-3} & 1.1153\times10^{-2} & 6.0768\times10^{-3} & 8.9177\times10^{-1} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 1.2005 & -1.6042\times10^{-1} & -7.2768\times10^{-1} \\ -9.9678\times10^{-2} & 1.3136 & -4.7075\times10^{-1} \\ 4.1474\times10^{-1} & 4.5015\times10^{-2} & 5.7579\times10^{-1} \\ 6.1379\times10^{-1} & 4.9669\times10^{-1} & 9.5778\times10^{-1} \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 5.8622\times10^{-1} & 2.3180\times10^{-1} \\ 0 & 0 & -1.4549\times10^{-1} & 9.3398\times10^{-1} \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

References & Bibliography

Alonge, F., D'Ippolito, F., Ferrante, G., and Raimondi, F.M. (1998). Parameter identification of induction motor model using genetic algorithms. *IEE Proceedings on Control Theory and Applications*, **145**, 6, 587–593.

Åström, K.J., and Hägglund, T. (1984). Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica*, **20**, 5, 645–651.

Åström, K.J., and Hägglund, T. (1995). *PID controllers: Theory, design, and tuning*. 2nd Edition, Instrument Society of America.

Åström, K.J., and Wittenmark, B. (1995). *Adaptive control*. 2nd Edition, Addison-Wesley.

Baker, J.E. (1985). Adaptive selection methods for genetic algorithms. In Grefenstette, J.J. (Ed.), *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Laurence Erlbaum Associates, pp. 101–111.

Baker, J.E. (1987). Reducing bias and inefficiency in the selection algorithm. In Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, pp. 14–21.

Beasley, D., Bull, D.R., and Martin, R.R. (1993). A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, **1**, 2, 101–125.

Bethke, A.D. (1981). *Genetic algorithms as function optimizers*. Doctoral Dissertation, University of Michigan (University Microfilms No. 81-06101).

Booker, L.B. (1982). *Intelligent behavior as an adaptation to the task environment*. Doctoral Dissertation, University of Michigan (University Microfilms No. 82-14966).

Booker, L.B. (1985). Improving the performance of genetic algorithms in classifier systems. In Grefenstette, J.J. (Ed.), *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Laurence Erlbaum Associates, pp. 80–92.

Caruana, R.A., Eshelman, L.J., and Schaffer, J.D. (1989). Representation and hidden bias II: Eliminating defining length bias in genetic search via shuffle crossover. In Sridharan, N.S. (Ed.), *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 750–755.

Caruana, R.A., and Schaffer, J.D. (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In Laird, L. (Ed.), *Proceedings of the Fifth International Workshop on Machine Learning*, Morgan Kaufmann, pp. 153–161.

Cavicchio, Jr., D.J. (1970). *Adaptive search using simulated evolution.* Doctoral Dissertation, University of Michigan.

Cavicchio, Jr., D.J. (1972). Reproductive adaptive plans. *Proceedings of the 1972 Annual Conference of the Association for Computing Machinery*, pp. 60–70.

Chen, B.-S., and Cheng, Y.-M. (1998). A structure-specified H^∞ optimal control design for practical applications: A genetic approach. *IEEE Transactions on Control Systems Technology*, **6**, 6, 707–718.

Chien, K.L., Hrones, J.A., and Reswick, J.B. (1952). On the automatic control of generalized passive systems. *Transactions of the American Society of Mechanical Engineers*, **74**, 175–185.

Chipperfield, A., and Fleming, P. (1995). Genetic algorithms in control systems engineering. *Control and Computers*, **23**, 3, 88–94.

Chipperfield, A., and Fleming, P. (1996). Multiobjective gas turbine engine controller design using genetic algorithms. *IEEE Transactions on Industrial Electronics*, **43**, 5, 583–587.

Clarke, T., and Davies, R. (1997). Robust eigenstructure assignment using the genetic algorithm and constrained state feedback. *Proceedings of the Institute of Mechanical Engineers*, **211**, 53–61.

Clarke, D.W., and Mohtadi, C. (1989). Properties of generalized predictive control. *Automatica*, **25**, 6, 859–875.

Clarke, D.W., Mohtadi, C., and Tuffs, P.S. (1987a). Generalized predictive control — Part I. The basic algorithm. *Automatica*, **23**, 2, 137–148.

Clarke, D.W., Mohtadi, C., and Tuffs, P.S. (1987b). Generalized predictive control — Part II. Extensions and interpretations. *Automatica*, **23**, 2, 149–160.

Cohen, G.H., and Coon, G.A. (1953). Theoretical consideration of retarded control. *Transactions of the American Society of Mechanical Engineers*, **75**, 827–834.

Culberson, J.C. (1998). On the futility of blind search: An algorithmic view of "no free lunch". *Evolutionary Computation*, **6**, 2, 109–127.

Cuthrell, J.E., Rivera, D.E., Schmidt, W.J., and Vegeais, J.A. (1990). Solution to the Shell standard control problem. *Proceedings of the Second Shell Process Control Workshop*, Butterworths, pp. 27–58.

Dan-Isa, A., and Atherton, D.P. (1997). Time-domain method for the design of optimal linear controllers. *IEE Proceedings on Control Theory and Applications*, **144**, 4, 287–292.

Dantzig, G.B. (1949). Programming of interdependent activities. II. Mathematical model. *Econometrica*, **17**, 200–211.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In Schaffer, J.D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 61–69.

Davis, L. (1991). *Handbook of genetic algorithms*. Van Nostrand Reinhold.

De Jong, K.A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Doctoral Dissertation, University of Michigan (University Microfilms No. 76-9381).

Deb, K., and Goldberg, D.E. (1989). An investigation of niche and species formation in genetic function optimization. In Schaffer, J.D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 42–50.

Edmunds, J.M. (1979). Control system design and analysis using closed-loop Nyquist and Bode arrays. *International Journal of Control*, **30**, 5, 773–802.

Filipič, B., Urbančič, T., and Križman, V. (1999). A combined machine learning and genetic algorithm approach to controller design. *Engineering Applications of Artificial Intelligence*, **12**, 4, 401–409.

Fletcher, R. (1987). *Practical methods of optimization*. 2nd Edition, John Wiley & Sons.

Fogel, D.B. (1994). An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, **5**, 1, 3–14.

Fonseca, C.M., and Fleming, P.J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 416–423.

Fonseca, C.M., and Fleming, P.J. (1994). Multiobjective optimal controller design with genetic algorithms. *Proceedings of the International Conference on Control '94*, IEE, pp. 745–749.

Fonseca, C.M., and Fleming, P.J. (1995). An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, **3**, 1, 1–16.

Fonseca, C.M., and Fleming, P.J. (1996). Non-linear system identification with multiobjective genetic algorithms. *Proceedings of the Thirteenth Triennial IFAC World Congress*, pp. 187–192.

Fonseca, C.M., and Fleming, P.J. (1998a). Multiobjective optimization and multiple constraint handling with evolutionary algorithms – Part I: A unified formulation. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, **28**, 1, 26–37.

Fonseca, C.M., and Fleming, P.J. (1998b). Multiobjective optimization and multiple constraint handling with evolutionary algorithms – Part II: Application example. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, **28**, 1, 38–47.

French, I.G., Cox, C.S., and Ho, C.K.S. (1997). Genetic algorithms in model structure and controller structure identification. *Proceedings of the Institute of Mechanical Engineers*, **211**, 333–343.

Gagnon, E., Pomerleau, A., and Desbiens, A. (1999). Mu-Synthesis of robust decentalised PI controllers. *IEE Proceedings on Control Theory and Applications*, **146**, 4, 289–294.

García, C.E., and Morshedi, A.M. (1986). Quadratic programming solution of Dynamic Matrix Control (QDMC). *Chemical Engineering Communications*, **46**, 73–87.

García, C.E., Prett, D.M., and Morari, M. (1989). Model predictive control: Theory and practice — A survey. *Automatica*, **25**, 3, 335–348.

Gill, P.E., Murray, W., and Wright, M.H. (1981). *Practical optimization*. Academic Press.

Goldberg, D.E. (1985). *Optimal initial population size for binary-coded genetic algorithms*. TCGA Report No. 85001, The Clearinghouse for Genetic Algorithms, University of Alabama, Tuscaloosa.

Goldberg, D.E. (1989a). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.

Goldberg, D.E. (1989b). Sizing populations in serial and parallel genetic algorithms. In Schaffer, J.D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 70–79.

Goldberg, D.E. (1994). Genetic and evolutionary algorithms come of age. *Communications of the Association for Computing Machinery*, **37**, 3, 113–119.

Goldberg, D.E., and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In Rawlins, G. (Ed.), *Foundations of Genetic Algorithms*, Morgan Kaufmann, pp. 69–93.

Goldberg, D.E., Deb, K., and Clark, J.H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, **6**, 333–362.

Goldberg, D.E., Deb, K., and Clark, J.H. (1993). Accounting for noise in the sizing of populations. In Whitley, L.D. (Ed.), *Foundations of Genetic Algorithms 2*, Morgan Kaufmann, pp. 127–140.

Goldberg, D.E., Deb, K., Kargupta, H., and Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 56–64.

Goldberg, D.E., Deb, K., and Korb, B. (1990). Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems*, **4**, 415–444.

Goldberg, D.E., Korb, B., and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, **3**, 493–530.

Goldberg, D.E., and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp. 41–49.

Gray, G.J., Murray-Smith, D.J., Li, Y., Sharman, K.C., and Weinbrenner, T. (1998). Nonlinear model structure identification using genetic programming. *Control Engineering Practice*, **6**, 11, 1341–1352.

Grefenstette, J.J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-16**, 1, 122–128.

Grefenstette, J.J. (1987). Incorporating problem specific knowledge into genetic algorithms. In Davis, L. (Ed.), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann, pp. 42–60.

Gürocak, H.B. (1999). A genetic-algorithm-based method for tuning fuzzy logic controllers. *Fuzzy Sets and Systems*, **108**, 1, 39–47.

Hägglund, T. (1999). On the stability of integral anti-windup loops in discrete-time PID controllers. *Personal communication by electronic mail*, 17 May 1999.

Halevi, Y., Palmor, Z.J., and Efrati, T. (1997). Automatic tuning of decentralized PID controllers for MIMO processes. *Journal of Process Control*, **7**, 2, 119–128.

Hancock, P.J.B. (1994). An empirical comparison of selection methods in evolutionary algorithms. In. Fogarty, T.C. (Ed.), *Evolutionary Computing: Artificial Intelligence and Simulation of Behaviour (AISB) Workshop, Leeds, UK, April 1994 – Selected Papers*, Springer-Verlag, pp. 80–94.

Hang, C.C., Loh, A.P., and Vasnani, V.U. (1994). Relay feedback auto-tuning of cascade controllers. *IEEE Transactions on Control Systems Technology*, **2**, 1, 42–45.

Hart, W.E., and Belew, R.K. (1996). Optimization with genetic algorithm hybrids that use local search. In Belew, R.K., and Mitchell, M. (Eds.), *Adaptive Individuals in Evolving Populations*, Addison-Wesley, pp. 483–496.

Herrera, F., Lozano, M., and Verdegay, J.L. (1998). A learning process for fuzzy control rules using genetic algorithms. *Fuzzy Sets and Systems*, **100**, 1–3, 143–158.

Hesser, J., and Männer, R. (1991). Towards an optimal mutation probability for genetic algorithms. In Schwefel, H.-P., and Männer, R. (Eds.), *Parallel Problem Solving from Nature*, Springer-Verlag, pp. 23–32.

Hillis, W.D. (1992). Co-evolving parasites improve simulated evolution as an optimization procedure. In Langton, C.G., Taylor, C., Farmer, J.D, and Rasmussen, S. (Eds.), *Artificial Life II*, Addison-Wesley, pp. 313–324.

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press (2nd Edition, MIT Press, 1992).

Holland, J.H. (1987). Genetic algorithms and classifier systems: Foundations and future directions. In Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp. 82–89.

Holland, J.H. (1992). Genetic algorithms. *Scientific American*, **267**, 1, 44–50.

Hong, X., and Billings, S.A. (1999). Parameter estimation based on stacked regression and evolutionary algorithms. *IEE Proceedings on Control Theory and Applications*, **146**, 5, 406–414.

Huang, W., and Lam, H.N. (1997). Using genetic algorithms to optimize controller parameters for HVAC systems. *Energy and Buildings*, **26**, 3, 277–282.

Hunt, K.J. (1992a). Polynomial LQG and H_∞ controller synthesis: A genetic algorithm solution. *Proceedings of the Thirty-First IEEE Conference on Decision and Control*, pp. 3604–3609.

Hunt, K.J. (1992b). Optimal control system synthesis with genetic algorithms. In Männer, R., and Manderick, B. (Eds.), *Parallel Problem Solving from Nature 2*, Elsevier Science, pp. 381–389.

Hwang, H.S. (1999). Automatic design of fuzzy rule base for modelling and control using evolutionary programming. *IEE Proceedings on Control Theory and Applications*, **146**, 1, 9–16.

Hwang, C.-L., and Masud, A.S.M. (1979). Multiple objective decision making – Methods and applications. *Lecture Notes in Economics and Mathematical Systems*, **164**, Springer-Verlag.

Jeon, Y.-S., Lee, C.-O., and Hong, Y.-S. (1998). Optimization of the control parameters of a pneumatic servo cylinder drive using genetic algorithms. *Control Engineering Practice*, **6**, 7, 847–853.

Jury, E.I. (1973). *Theory and application of the z-Transform method*. Robert E. Krieger Publishing Company.

Kawabe, T., Tagami, T., and Katayama, T. (1996). A genetic algorithm based minimax optimal design of robust I-PD controller. *Proceedings of the International Conference on Control '96*, IEE, pp. 436–441.

Kawabe, T., Tagami, T., and Okamura, M. (1998). A matrix inequality design of robust PID controller with two degrees of freedom. *Transactions of the Institute of Systems, Control, and Information Engineers*, **11**, 1, 19–25.

Koza, J.R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. MIT Press.

Krishnakumar, K., and Goldberg, D.E. (1992). Control system optimization using genetic algorithms. *Journal of Guidance, Control, and Dynamics*, **15**, 3, 735–740.

Kristinsson, K., and Dumont, G.A. (1992). System identification and control using genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, **22**, 5, 1033–1046.

Lennon, W.K., and Passino, K.M. (1999). Genetic adaptive identification and control. *Engineering Applications of Artificial Intelligence*, **12**, 2, 185–200.

Li, C.J., and Jeon, Y.C. (1993). Genetic algorithm in identifying non linear auto regressive with exogenous input models for non linear systems. *Proceedings of the 1993 American Control Conference*, pp. 2305–2309.

Li, C.J., Tzeng, T.-C., and Jeon, Y.C. (1997). A learning controller based on a nonlinear ARX inverse model identified by a genetic algorithm. *International Journal of Systems Science*, **28**, 8, 847–855.

Lian, S.T., Marzuki, K., and Rubiyah, Y. (1998). Tuning of a neuro-fuzzy controller by genetic algorithms with an application to a coupled-tank liquid-level control system. *Engineering Applications of Artificial Intelligence*, **11**, 4, 517–529.

Linkens, D.A., and Nyongesa, H.O. (1995a). Genetic algorithms for fuzzy control. Part 1: Offline system development and application. *IEE Proceedings on Control Theory and Applications*, **142**, 3, 161–176.

Linkens, D.A., and Nyongesa, H.O. (1995b). Genetic algorithms for fuzzy control. Part 2: Online system development and application. *IEE Proceedings on Control Theory and Applications*, **142**, 3, 177–185.

Linkens, D.A., and Nyongesa, H.O. (1996a). A hierarchical multivariable fuzzy controller for learning with genetic algorithms. *International Journal of Control*, **63**, 5, 865–883.

Linkens, D.A., and Nyongesa, H.O. (1996b). Learning systems in intelligent control: An appraisal of fuzzy, neural and genetic algorithm control applications. *IEE Proceedings on Control Theory and Applications*, **143**, 4, 367–386.

Liu, G.P., and Kadirkamanathan, V. (1999). Multiobjective criteria for neural network structure selection and identification of nonlinear systems using genetic algorithms. *IEE Proceedings on Control Theory and Applications*, **146**, 5, 373–382.

Loh, A.P., Tan, W.W., and Vasnani, V.U. (1994). Relay feedback of multivariable systems and its use for auto-tuning of multi-loop PI controllers. *Proceedings of the International Conference on Control '94*, IEE, pp. 1049–1054.

Loh, A.P., and Vasnani, V.U. (1994). Necessary conditions for limit cycles in multiloop relay systems. *IEE Proceedings on Control Theory and Applications*, **141**, 3, 163–168.

Lopez, A.M., Miller, J.A., Smith, C.L., and Murrill, P.W. (1967). Tuning controllers with error-integral criteria. *Instrumentation Technology*, November Issue, 57–62.

Maciejowski, J.M. (1994). Robustness of multivariable Smith predictors. *Journal of Process Control*, **4**, 1, 29–32.

Macready, W.G., and Wolpert, D.H. (1996). What makes an optimization problem hard? *Complexity*, **1**, 5, 40–46.

Macready, W.G., and Wolpert, D.H. (1998). Bandit problems and the exploration/exploitation tradeoff. *IEEE Transactions on Evolutionary Computation*, **2**, 1, 2–22.

Marrison, C.I., and Stengel, R.F. (1997). Robust control system design using random search and genetic algorithms. *IEEE Transactions on Automatic Control*, **42**, 6, 835–839.

Martinez, M., Senent, J.S., and Blasco, X. (1998). Generalized predictive control using genetic algorithms (GAGPC). *Engineering Applications of Artificial Intelligence*, **11**, 3, 355–367.

McKay, B., Willis, M., and Barton, G. (1997). Steady-state modelling of chemical process systems using genetic programming. *Computers and Chemical Engineering*, **21**, 9, 981–996.

Mitchell, M. (1996). *An introduction to genetic algorithms*. MIT Press.

Nelder, J.A., and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, **7**, 308–313.

Onnen, C., Babuška, R., Kaymak, U., Sousa, J.M., Verbruggen, H.B., and Isermann, R. (1997). Genetic algorithms for optimization in predictive control. *Control Engineering Practice*, **5**, 10, 1363–1372.

Palmor, Z.J., Halevi, Y., and Krasney, N. (1995). Automatic tuning of decentralized PID controllers for TITO processes. *Automatica*, **31**, 7, 1001–1010.

Patton, R.J., Chen, J., and Liu, G.P. (1997). Robust fault detection of dynamic systems via genetic algorithms. *Proceedings of the Institute of Mechanical Engineers*, **211**, 357–364.

Patton, R.J., and Liu, G.P. (1994). Robust control design via eigenstructure assignment, genetic algorithms and gradient-based optimisation. *IEE Proceedings on Control Theory and Applications*, **141**, 3, 202–208.

Pham, D.T., and Karaboga, D. (1997). Genetic algorithms with variable mutation rates: Application to fuzzy logic controller design. *Proceedings of the Institute of Mechanical Engineers*, **211**, 157–167.

Pham, Q.T. (1998). Dynamic optimization of chemical engineering processes by an evolutionary method. *Computers and Chemical Engineering*, **22**, 7–8, 1089–1097.

Porter, B., and Jones, A.H. (1992). Genetic tuning of digital PID controllers. *Electronics Letters*, **28**, 9, 843–844.

Powell, M.J.D. (1983). Variable metric methods for constrained optimization. In Bachem, A., Grotschel, M., and Korte, B. (Eds.), *Mathematical Programming: The State of the Art*, Springer-Verlag, pp. 288–311.

Prett, D.M., García, C.E., and Morari, M. (1990). *The Second Shell Process Control Workshop*. Butterworths.

Prett, D.M., and Morari, M. (1987). *The Shell Process Control Workshop*. Butterworths.

Rahmoun, A., and Benmohamed, M. (1998). Genetic algorithm based methodology to generate automatically optimal fuzzy rules. *IEE Proceedings on Control Theory and Applications*, **145**, 6, 583–586.

Rao, S.S. (1996). *Engineering optimization: Theory and practice*. 3rd Edition, John Wiley & Sons.

Richardson, J.T., Palmer, M.R., Liepins, G., and Hilliard, M. (1989). Some guidelines for genetic algorithms with penalty functions. In Schaffer, J.D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 191–197.

Rodriguez-Vazquez, K., and Fleming, P.J. (1997). A genetic programming/NARMAX approach to nonlinear system identification. *Proceedings of the Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, IEE, pp. 409–414.

Rodriguez-Vazquez, K., and Fleming, P.J. (1998). Multi-objective genetic programming for nonlinear system identification. *Electronics Letters*, **34**, 9, 930–931.

Ros, H. (1989). Some results on Boolean concept learning by genetic algorithms. In Schaffer, J.D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 28–33.

Rudolph, G. (1994). Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, **5**, 1, 96–101.

Salomon, R. (1998). Evolutionary algorithms and gradient search: Similarities and differences. *IEEE Transactions on Evolutionary Computation*, **2**, 2, 45–55.

Santos, A., and Dourado, A. (1999). Global optimization of energy and production in process industries: A genetic algorithm application. *Control Engineering Practice*, **7**, 4, 549–554.

Schaffer, J.D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In Grefenstette, J.J. (Ed.), *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Laurence Erlbaum Associates, pp. 93–100.

Schaffer, J.D. (1987). An adaptive crossover distribution mechanism for genetic algorithms. In Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp. 36–40.

Schaffer, J.D., Caruana, R.A., Eshelman, L.J., and Das, R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In Schaffer, J.D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 51–60.

Semino, D., and Scali, C. (1998). Improved identification and autotuning of PI controllers for MIMO processes by relay techniques. *Journal of Process Control*, **8**, 3, 219–227.

Shafiei, Z., and Shenton, A.T. (1997). Frequency-domain design of PID controllers for stable and unstable systems with time delay. *Automatica*, **33**, 12, 2223–2232.

Shieh, L.S., Wang, W., and Tsai, J.S.H. (1999). Optimal digital design of hybrid uncertain systems using genetic algorithms. *IEE Proceedings on Control Theory and Applications*, **146**, 2, 119–130.

Shieh, L.S., Zheng, J., and Wang, W. (1997). Digital modeling and digital redesign of analog uncertain systems using genetic algorithms. *Journal of Guidance, Control, and Dynamics*, **20**, 4, 721–728.

Smith, O.J.M. (1957). Closer control of loops with dead time. *Chemical Engineering Progress*, **53**, 5, 217–219.

Spears, W.M. (1993). Crossover or mutation? In Whitley, L.D. (Ed.), *Foundations of Genetic Algorithms 2*, Morgan Kaufmann, pp. 221–237.

Stadnyk, I. (1987). Schema recombination in a pattern recognition problem. In Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, Laurence Erlbaum Associates, pp. 27–35.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In Schaffer, J.D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 2–9.

Tan, K.C., and Li, Y. (1997). Evolutionary system identification in the time domain. *Proceedings of the Institute of Mechanical Engineers*, **211**, 319–323.

Takahashi, R.H.C., Peres, P.L.D., and Ferreira, P.A.V. (1997). Multiobjective H_2/H_∞ guaranteed cost PID design. *IEEE Control Systems Magazine*, **17**, 5, 37–47.

Thierens, D. (1995). *Analysis and design of genetic algorithms*. Doctoral Dissertation, Katholieke Universiteit Leuven, Belgium.

Thierens, D., and Goldberg, D.E. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 38–45.

Thierens, D., and Goldberg, D.E. (1994). Convergence models of genetic algorithm selection schemes. In Davidor, Y., Schwefel, H.-P., and Männer, R. (Eds.), *Parallel Problem Solving from Nature 3*, Springer-Verlag, pp. 119–129.

Thompson, H.A., Chipperfield, A.J., Fleming, P.J., and Legge, C. (1999). Distributed aero-engine control systems architecture selection using multi-objective optimisation. *Control Engineering Practice*, **7**, 5, 655–664.

Trebi-Ollennu, A., and White, B.A. (1997). Multiobjective fuzzy genetic algorithm optimisation approach to nonlinear control system design. *IEE Proceedings on Control Theory and Applications*, **144**, 2, 137–142.

Varšek, A., Urbančič, T., and Filipič, B. (1993). Genetic algorithms in controller design and tuning. *IEEE Transactions on Systems, Man, and Cybernetics*, **23**, 5, 1330–1339.

Vlachos, C., Evans, J.T., and Williams, D. (1997). PI controller tuning for multivariable processes using genetic algorithms. *Proceedings of the Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, IEE, pp. 43–49.

Vlachos, C., Williams, D., and Gomm, J.B. (1998). Solution to the Shell standard control problem using genetic algorithms. *Proceedings of the International Conference on Control '98*, IEE, pp. 1587–1592.

Vlachos, C., Williams, D., and Gomm, J.B. (1999a). Genetic approach to decentralised PI controller tuning for multivariable processes. *IEE Proceedings on Control Theory and Applications*, **146**, 1, 58–64.

Vlachos, C., Williams, D., and Gomm, J.B. (1999b). Tuning decentralised controllers for multivariable systems – A genetic approach. *Proceedings of the Second Wismar Symposium on Automatic Control*, Hochschule Wismar, Germany.

Wang, P., and Kwok, D.P. (1994). Optimal design of PID process controllers based on genetic algorithms. *Control Engineering Practice*, **2**, 4, 641–648.

Warwick, K., and Kang, Y.-H. (1998). Self-tuning proportional, integral and derivative controller based on genetic algorithm least squares. *Proceedings of the Institute of Mechanical Engineers*, **212**, 437–448.

Wolpert, D.H., and Macready, W.G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, **1**, 1, 67–82.

Wong, C.-C., and Fan, C.-S. (1999). Rule mapping fuzzy controller design. *Fuzzy Sets and Systems*, **108**, 3, 253–261.

Wood, M.K., and Dantzig, G.B. (1949). Programming of interdependent activities. I. General discussion. *Econometrica*, **17**, 193–199.

Wood, R.K., and Berry, M.W. (1973). Terminal composition control of a binary distillation column. *Chemical Engineering Science*, **28**, 1707–1717.

Zalzala, A.M.S., and Fleming, P.J. (1996). Genetic algorithms: Principles and applications in engineering systems. *Neural Network World*, **6**, 5, 803–820.

Zhou, K., Doyle, J.C., and Glover, K. (1996). *Robust and optimal control*. Prentice-Hall.

Zhuang, M., and Atherton, D.P. (1994). PID controller design of a TITO system. *IEE Proceedings on Control Theory and Applications*, **141**, 2, 111–120.

Ziegler, J.G., and Nichols, N.B. (1942). Optimum settings for automatic controllers. *Transactions of the American Society of Mechanical Engineers*, **64**, 759–768.

Zuo, W. (1995). Multivariable adaptive control for a space station using genetic algorithms. *IEE Proceedings on Control Theory and Applications*, **142**, 2, 81–87.

Zuo, W. (1997). A genetic approach to adaptive control system design. *Proceedings of the Institute of Mechanical Engineers*, **211**, 15–23.

Published Work

Vlachos, C., Evans, J.T., and Williams, D. (1997). PI controller tuning for multivariable processes using genetic algorithms. *Proceedings of the Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, IEE, pp. 43–49.

Vlachos, C., Williams, D., and Gomm, J.B. (1998). Solution to the Shell standard control problem using genetic algorithms. *Proceedings of the International Conference on Control '98*, IEE, pp. 1587–1592.

Vlachos, C., Williams, D., and Gomm, J.B. (1999a). Genetic approach to decentralised PI controller tuning for multivariable processes. *IEE Proceedings on Control Theory and Applications*, **146**, 1, 58–64.

Vlachos, C., Williams, D., and Gomm, J.B. (1999b). Tuning decentralised controllers for multivariable systems – A genetic approach. *Proceedings of the Second Wismar Symposium on Automatic Control*, Hochschule Wismar, Germany.