Ernest Orlando Lawrence
Berkeley National Laboratory

GenOpt(R)
Generic Optimization Program

User Manual
Version 3.0.0β1

Simulation Research Group
Building Technologies Department
Environmental Energy Technologies Division
Lawrence Berkeley National Laboratory
Berkeley, CA 94720

http://SimulationResearch.lbl.gov

Michael Wetter
MWetter@lbl.gov

February 20, 2009

Notice:

This work was supported by the U.S. Department of Energy (DOE), by the Swiss Academy of Engineering Sciences (SATW), and by the Swiss National Energy Fund (NEFF).

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory), subject to receipt of any required approvals from U.S. Department of Energy.
All rights reserved.

Contents

1 Abstract **5**

2 Notation **6**

3 Introduction **7**

4 Optimization Problems **10**
4.1 Classification of Optimization Problems 10
4.1.1 Problems with Continuous Variables 10
4.1.2 Problems with Discrete Variables 10
4.1.3 Problems with Continuous and Discrete Variables 11
4.1.4 Problems that use a Building Simulation Program 11
4.2 Algorithm Selection 12
4.2.1 Problem $\mathbf{P}_c$ with $n > 1$ 12
4.2.2 Problem $\mathbf{P}_{cg}$ with $n > 1$ 13
4.2.3 Problem $\mathbf{P}_c$ with $n = 1$ 14
4.2.4 Problem $\mathbf{P}_{cg}$ with $n = 1$ 14
4.2.5 Problem $\mathbf{P}_d$ 14
4.2.6 Problem $\mathbf{P}_{cd}$ and $\mathbf{P}_{cdg}$ 14
4.2.7 Functions with Several Local Minima 14

5 Algorithms for Multi-Dimensional Optimization **15**
5.1 Generalized Pattern Search Methods (Analysis) 15
5.1.1 Assumptions 16
5.1.2 Characterization of GPS Algorithms 17
5.1.3 Model Adaptive Precision GPS Algorithm 18
5.1.4 Convergence Results 19
a) Unconstrained Minimization 19
b) Box-Constrained Minimization 20
5.2 Generalized Pattern Search Methods (Implementations) . 20
5.2.1 Coordinate Search Algorithm 21
a) Algorithm Parameters 21
b) Global Search 21
c) Local Search 21
d) Parameter Update 22
e) Keywords 22
5.2.2 Hooke-Jeeves Algorithm 23
a) Algorithm Parameters 23
b) Map for Exploratory Moves 23

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

c) Global Search Set Map 23
d) Local Search Direction Map 24
e) Parameter Update 24
f) Keywords 24
5.2.3 Multi-Start GPS Algorithms 25
5.3 Discrete Armijo Gradient 27
5.3.1 Keywords 29
5.4 Particle Swarm Optimization 31
5.4.1 PSO for Continuous Variables 31
a) Neighborhood Topology 32
b) Model PSO Algorithm 34
c) Particle Update Equation 34
(i) Inertia Weight 34
(ii) Constriction Coefficient 35
5.4.2 PSO for Discrete Variables 36
5.4.3 PSO for Continuous and Discrete Variables 37
5.4.4 PSO on a Mesh 37
5.4.5 Population Size and Number of Generations 37
5.4.6 Keywords 38
5.5 Hybrid GPS Algorithm with PSO Algorithm 41
5.5.1 For Continuous Variables 41
5.5.2 For Continuous and Discrete Variables 42
5.5.3 Keywords 42
5.6 Simplex Algorithm of Nelder and Mead 44
5.6.1 Main Operations 44
5.6.2 Basic Algorithm 46
5.6.3 Stopping Criteria 48
5.6.4 O'Neill's Modification 49
5.6.5 Modification of Stopping Criteria 49
5.6.6 Benchmark Tests 51
5.6.7 Keywords 54

6 Algorithms for One-Dimensional Optimization 55
6.1 Interval Division Algorithms 55
6.1.1 General Interval Division 55
6.1.2 Golden Section Interval Division 56
6.1.3 Fibonacci Division 57
6.1.4 Comparison of Efficiency 58
6.1.5 Master Algorithm for Interval Division 58
6.1.6 Keywords 59

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

7 Algorithms for Parametric Runs **61**
7.1 Parametric Runs by Single Variation 61
7.1.1 Algorithm Description 61
7.1.2 Keywords 62
7.2 Parametric Runs on a Mesh 62
7.2.1 Algorithm Description 62
7.2.2 Keywords 63

8 Constraints **64**
8.1 Constraints on Independent Variables 64
8.1.1 Box Constraints 64
8.1.2 Coupled Linear Constraints 65
8.2 Constraints on Dependent Variables 65
8.2.1 Barrier Functions 66
8.2.2 Penalty Functions 66
8.2.3 Implementation of Barrier and Penalty Functions . 67

9 Program **68**
9.1 Interface to the Simulation Program 68
9.2 Interface to the Optimization Algorithm 69
9.3 Package `genopt.algorithm` 69
9.4 Implementing a New Optimization Algorithm 71

10 Installing and Running GenOpt **73**
10.1 System Requirements 73
10.2 Installing and uninstalling GenOpt 73
10.3 Running GenOpt 73
10.3.1 Running GenOpt from the file explorer 73
10.3.2 Running GenOpt from the command line 73

11 Setting Up an Optimization Problem **75**
11.1 File Specification 75
11.1.1 Initialization File 76
11.1.2 Configuration File 81
11.1.3 Command File 83
a) Specification of a Continuous Parameter . 83
b) Specification of a Discrete Parameter . . 84
c) Specification of Input Function Objects . 85
d) Structure of the Command File 86
11.1.4 Log File 87
11.1.5 Output Files 87
11.2 Resolving Directory Names for Parallel Computing 87

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

11.3 Pre-Processing and Post-Processing 89
a) Function Objects 89
b) Pre-Processing 90
c) Post-Processing 90
11.4 Truncation of Digits of the Cost Function Value 91

12 Conclusion **93**

13 Acknowledgment **94**

14 Legal **95**
14.1 Copyright Notice 95
14.2 License agreement 95

A Benchmark Tests **97**
A.1 Rosenbrock 97
A.2 Function 2D1 98
A.3 Function Quad 99

Product and company names mentioned herein may be the trademarks of their respective owners. Any rights not expressly granted herein are reserved.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

1 Abstract

GenOpt is an optimization program for the minimization of a cost function that is evaluated by an external simulation program. It has been developed for optimization problems where the cost function is computationally expensive and its derivatives are not available or may not even exist. GenOpt can be coupled to any simulation program that reads its input from text files and writes its output to text files. The independent variables can be continuous variables (possibly with lower and upper bounds), discrete variables, or both, continuous and discrete variables. Constraints on dependent variables can be implemented using penalty or barrier functions. GenOpt uses parallel computing to evaluate the simulations.

GenOpt has a library with local and global multi-dimensional and one-dimensional optimization algorithms, and algorithms for doing parametric runs. An algorithm interface allows adding new minimization algorithms without knowing the details of the program structure.

GenOpt is written in Java so that it is platform independent. The platform independence and the general interface make GenOpt applicable to a wide range of optimization problems.

GenOpt has not been designed for linear programming problems, quadratic programming problems, and problems where the gradient of the cost function is available. For such problems, as well as for other problems, special tailored software exists that is more efficient.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

2 Notation

1. We use the notation $a \triangleq b$ to denote that a is equal to b by definition. We use the notation $a \leftarrow b$ to denote that a is assigned the value of b.

2. $\mathbb{R}^n$ denotes the Euclidean space of n-tuplets of real numbers. Vectors $x \in \mathbb{R}^n$ are always column vectors, and their elements are denoted by superscripts. The inner product in $\mathbb{R}^n$ is denoted by $\langle \cdot, \cdot \rangle$ and for $x, y \in \mathbb{R}^n$ defined by $\langle x, y \rangle \triangleq \sum_{i=1}^{n} x^i \, y^i$. The norm in $\mathbb{R}^n$ is denoted by $\| \cdot \|$ and for $x \in \mathbb{R}^n$ defined by $\|x\| \triangleq \langle x, x \rangle^{1/2}$.

3. We denote by $\mathbb{Z}$ the set of integers, by $\mathbb{Q}$ the set of rational numbers, and by $\mathbb{N} \triangleq \{0, 1, \ldots\}$ the set of natural numbers. The set $\mathbb{N}_+$ is defined as $\mathbb{N}_+ \triangleq \{1, 2, \ldots\}$. Similarly, vectors in $\mathbb{R}^n$ with strictly positive elements are denoted by $\mathbb{R}^n_+ \triangleq \{x \in \mathbb{R}^n \mid x^i > 0, \; i \in \{1, \ldots, n\}\}$ and the set $\mathbb{Q}_+$ is defined as $\mathbb{Q}_+ \triangleq \{q \in \mathbb{Q} \mid q > 0\}$.

4. Let $\mathbb{W}$ be a set containing a sequence $\{w_i\}_{i=0}^{k}$. Then, we denote by $\underline{w}_k$ the sequence $\{w_i\}_{i=0}^{k}$ and by $\underline{\mathbf{W}}_k$ the set of all $k+1$ element sequences in $\mathbb{W}$.

5. If $\mathbf{A}$ and $\mathbf{B}$ are sets, we denote by $\mathbf{A} \cup \mathbf{B}$ the union of $\mathbf{A}$ and $\mathbf{B}$ and by $\mathbf{A} \cap \mathbf{B}$ the intersection of $\mathbf{A}$ and $\mathbf{B}$.

6. If $\mathbf{S}$ is a set, we denote by $\overline{\mathbf{S}}$ the closure of $\mathbf{S}$ and by $2^{\mathbf{S}}$ the set of all nonempty subsets of $\mathbf{S}$.

7. If $\widehat{D} \in \mathbb{Q}^{n \times q}$ is a matrix, we will use the notation $\widehat{d} \in \widehat{D}$ to denote the fact that $\widehat{d} \in \mathbb{Q}^n$ is a column vector of the matrix $\widehat{D}$. Similarly, by $D \subset \widehat{D}$ we mean that $D \in \mathbb{Q}^{n \times p}$ $(1 \leq p \leq q)$ is a matrix containing only columns of $\widehat{D}$. Further, $\mathrm{card}(D)$ denotes the number of columns of D.

8. $f(\cdot)$ denotes a function where $(\cdot)$ stands for the undesignated variables. $f(x)$ denotes the value of $f(\cdot)$ at the point x. $f \colon A \to B$ indicates that the domain of $f(\cdot)$ is in the space A and its range in the space B.

9. We say that a function $f : \mathbb{R}^n \to \mathbb{R}$ is once continuously differentiable if $f(\cdot)$ is defined on $\mathbb{R}^n$, and if $f(\cdot)$ has continuous derivatives on $\mathbb{R}^n$.

10. For $x^* \in \mathbb{R}^n$ and $f \colon \mathbb{R}^n \to \mathbb{R}$ continuously differentiable, we say that x^* is stationary if $\nabla f(x^*) = 0$.

11. We denote by $\{e_i\}_{i=1}^{n}$ the unit vectors in $\mathbb{R}^n$.

12. We denote by $\rho \sim U(0, 1)$ that $\rho \in \mathbb{R}$ is a uniformly distributed random number, with $0 \leq \rho \leq 1$.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

3 Introduction

The use of system simulation for analyzing complex engineering problems is increasing. Such problems typically involve many independent variables[1], and can only be optimized by means of numerical optimization. Many designers use parametric studies to achieve better performance of such systems, even though such studies typically yield only partial improvement while requiring high labor time. In such parametric studies, one usually fixes all but one variable and tries to optimize a cost function[2] with respect to the non-fixed variable. The procedure is repeated iteratively by varying another variable. However, every time a variable is varied, all other variables typically become non-optimal and hence need also to be adjusted. It is clear that such a manual procedure is very time-consuming and often impractical for more than two or three independent variables.

GenOpt, a generic optimization program, has been developed to find with less labor time the independent variables that yield better performance of such systems. GenOpt does optimization of a user-supplied cost function, using a user-selected optimization algorithm.

In the most general form, the optimization problems addressed by GenOpt can be stated as follows: Let $\mathbf{X}$ be a user-specified constraint set, and let $f\colon \mathbf{X} \to \mathbb{R}$ be a user-defined cost function that is bounded from below. The constraint set $\mathbf{X}$ consists of all possible design options, and the cost function $f(\cdot)$ measures the system performance. GenOpt tries to find a solution to the problem[3]

$$\min_{x \in \mathbf{X}} f(x). \tag{3.1}$$

This problem is usually "solved" by iterative methods, which construct infinite sequences, of progressively better approximations to a "solution", i.e., a point that satisfies an optimality condition. If $\mathbf{X} \subset \mathbb{R}^n$, with some $n \in \mathbb{N}$, and $\mathbf{X}$ or $f(\cdot)$ is not convex, we do not have a test for global optimality, and the most one can obtain is a point that satisfies a local optimality condition. Furthermore, for $\mathbf{X} \subset \mathbb{R}^n$, tests for optimality are based on differentiability assumptions of the cost function. Consequently, optimization algorithms can fail, possibly far from a solution, if $f(\cdot)$ is not differentiable in the continuous independent variables. Some optimization algorithms are more likely to fail at

[1]The independent variables are the variables that are varied by the optimization algorithm from one iteration to the next. They are also called design parameters or free parameters.

[2]The cost function is the function being optimized. The cost function measures a quantity that should be minimized, such as a building's annual operation cost, a system's energy consumption, or a norm between simulated and measured values in a data fitting process. The cost function is also called objective function.

[3]If $f(\cdot)$ is discontinuous, it may only have an infimum (i.e., a greatest lower bound) but no minimum even if the constraint set $\mathbf{X}$ is compact. Thus, to be correct, (3.1) should be replaced by $\inf_{x \in \mathbf{X}} f(x)$. For simplicity, we will not make this distinction.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

discontinuities than others. GenOpt has algorithms that are not very sensitive to (small) discontinuities in the cost function, such as Generalized Pattern Search algorithms, which can also be used in conjunction with heuristic global optimization algorithms.

Since one of GenOpt's main application fields is building energy use or operation cost optimization, GenOpt has been designed such that it addresses the special properties of optimization problems in this area. In particular, GenOpt is designed for optimization problems with the following properties:

1. The cost function may have to be defined on approximate numerical solutions of differential algebraic equations, which may fail to be continuous (see Section 4.1.4).
2. The number of independent variables is small.[4]
3. Evaluating the cost function requires much more computation time than determining the values for the next iterate.
4. Analytical properties of the cost function (such as formula for the gradient) are not available.

GenOpt has the following properties:

1. GenOpt can be coupled to any simulation program that calculates the cost function without having to modify or recompile either program, provided that the simulation program reads its input from text files and writes its output to text files.
2. The user can select an optimization algorithm from an algorithm library, or implement a custom algorithm without having to recompile and understand the whole optimization environment.
3. GenOpt does not require an expression for the gradient of the cost function.

With GenOpt, it is easy to couple a new simulation program, specify the optimization variables and minimize the cost function. Therefore, in designing complex systems, as well as in system analysis, a generic optimization program like GenOpt offers valuable assistance. Note, however, that optimization is not easy: The efficiency and success of an optimization is strongly affected by the properties and the formulation of the cost function, and by the selection of an appropriate optimization algorithm.

This manual is structured as follows: In Section 4, we classify optimization problems and discuss which of GenOpt's algorithms can be used for each of these problems. Next, we explain the algorithms that are implemented in GenOpt: In Section 5, we discuss the algorithms for multi-dimensional optimization; in Section 6 the algorithms for one-dimensional optimization; and

[4]By small, we mean on the order of 10, but the maximum number of independent variables is not restricted in GenOpt.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

in Section 7 the algorithms for parametric runs. In Section 8, we discuss how constraints on independent variables are implemented, and how constraints on dependent variables can be implemented. In Section 9, we explain the structure of the GenOpt software, the interface for the simulation program and the interface for the optimization algorithms. How to install and start GenOpt is described in Section 10. Section 11 shows how to set up the configuration and input files, and how to use GenOpt's pre- and post-processing capabilities.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

4 Optimization Problems

4.1 Classification of Optimization Problems

We will now classify some optimization problems that can be solved with GenOpt's optimization algorithms. The classification will be used in Section 4.2 to recommend suitable optimization algorithms.

We distinguish between problems whose design parameters are continuous variables[1], discrete variables[2], or both. In addition, we distinguish between problems with and without inequality constraints on the dependent variables.

4.1.1 Problems with Continuous Variables

To denote box-constraints on independent continuous variables, we will use the notation

$$\mathbf{X} \triangleq \left\{ x \in \mathbb{R}^n \mid l^i \leq x^i \leq u^i,\ i \in \{1, \ldots, n\} \right\}, \tag{4.1}$$

where $-\infty \leq l^i < u^i \leq \infty$ for $i \in \{1, \ldots, n\}$.

We will consider optimization problems of the form

$$\mathbf{P}_c \qquad \min_{x \in \mathbf{X}} f(x), \tag{4.2}$$

where $f \colon \mathbb{R}^n \to \mathbb{R}$ is a once continuously differentiable cost function.

Now, we add inequality constraints on the dependent variables to (4.2) and obtain

$$\mathbf{P}_{cg} \qquad \min_{x \in \mathbf{X}} f(x), \tag{4.3a}$$

$$g(x) \leq 0, \tag{4.3b}$$

where everything is as in (4.2) and, in addition, $g \colon \mathbb{R}^n \to \mathbb{R}^m$ is a once continuously differentiable constraint function (for some $m \in \mathbb{N}$). We will assume that there exists an $x^* \in \mathbf{X}$ that satisfies $g(x^*) < 0$.

4.1.2 Problems with Discrete Variables

Next, we will discuss the situation where all design parameters can only take on user-specified discrete values.

Let $\mathbf{X}_d \subset \mathbb{Z}^{n_d}$ denote the constraint set with a finite, non-zero number of integers for each variable.

[1] Continuous variables can take on any value on the real line, possibly between lower and upper bounds.

[2] Discrete variables can take on only integer values.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

We will consider integer programming problems of the form

$$\mathbf{P}_d \qquad \min_{x \in \mathbf{X}_d} f(x). \tag{4.4}$$

4.1.3 Problems with Continuous and Discrete Variables

Next, we will allow for continuous and discrete independent variables.

We will use the notation

$$\mathbf{X} \triangleq \mathbf{X}_c \times \mathbf{X}_d, \tag{4.5a}$$
$$\mathbf{X}_c \triangleq \left\{x \in \mathbb{R}^{n_c} \mid l^i \leq x^i \leq u^i,\ i \in \{1, \ldots, n_c\}\right\}, \tag{4.5b}$$

where the bounds on the continuous independent variables satisfy $-\infty \leq l^i < u^i \leq \infty$ for $i \in \{1, \ldots, n_c\}$, and the constraint set $\mathbf{X}_d \subset \mathbb{Z}^{n_d}$ for the discrete variables is a user-specified set with a finite, non-zero number of integers for each variable.

We will consider mixed-integer programming problems of the form

$$\mathbf{P}_{cd} \qquad \min_{x \in \mathbf{X}} f(x), \tag{4.6a}$$
$$\tag{4.6b}$$

where $x \triangleq (x_c, x_d) \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}$, $f \colon \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} \to \mathbb{R}$ and $\mathbf{X}$ is as in (4.5).

Now, we add inequality constraints on the dependent variables to (4.6) and obtain

$$\mathbf{P}_{cdg} \qquad \min_{x \in \mathbf{X}} f(x), \tag{4.7a}$$
$$g(x) \leq 0, \tag{4.7b}$$

where everything is as in (4.6) and in addition $g \colon \mathbb{R}^{n_c} \times \mathbb{R}^{n_d} \to \mathbb{R}^m$ (for some $m \in \mathbb{N}$). We will assume that there exists an $x^* \in \mathbf{X}$ that satisfies $g(x^*) < 0$.

4.1.4 Problems whose Cost Function is Evaluated by a Building Simulation Program

Next, we will discuss problem $\mathbf{P}_c$ defined in (4.2) for the situation where the cost function $f \colon \mathbb{R}^n \to \mathbb{R}$ cannot be evaluated, but can be approximated numerically by approximating cost functions $f^* \colon \mathbb{R}^p_+ \times \mathbb{R}^n \to \mathbb{R}$, where the first argument is the precision parameter of the numerical solvers. This is typically the case when the cost is computed by a thermal building simulation program, such as EnergyPlus [CLW+01], TRNSYS [KDB76], or DOE-2 [WBB+93]. In such programs, computing the cost involves solving a system of partial and ordinary differential equations that are coupled to algebraic equations. In general, one cannot obtain an exact solution, but one can obtain an approximate numerical solution. Hence, the cost function $f(x)$ can only be approximated by an approximating cost function $f^*(\epsilon, x)$, where $\epsilon \in \mathbb{R}^q_+$ is a vector that contains

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

precision parameters of the numerical solvers. Consequently, the optimization algorithm can only be applied to $f^*(\epsilon, x)$ and not to $f(x)$.

In such thermal building simulation programs it is common that the termination criteria of the solvers that are used to solve the partial differential equations, ordinary differential equations, and algebraic equations depend on the independent variable x. Therefore, a perturbation of x can cause a change in the sequence of solver iterations, which causes the approximating cost functions $f^*(\epsilon, x)$ to be discontinuous in x. Furthermore, if variable step size integration methods are used, then the integration mesh can change from one simulation to the next. Therefore, part of the change in function values between different points is caused by a change of the number of solver iterations, and by a change of the integration mesh. Consequently, $f^*(\epsilon, \cdot)$ is discontinuous, and a descent direction for $f^*(\epsilon, \cdot)$ may not be a descent direction for $f(\cdot)$. Therefore, optimization algorithms can terminate at points that are non-optimal.

The best one can do in trying to solve optimization problems where the cost and constraint functions are evaluated by a thermal building simulation program that does not allow controlling the approximation error is to find points that are close to a local minimizer of $f(\cdot)$. Numerical experiments show that by using tight enough precision and starting the optimization algorithm with coarse initial values, one often comes close to a minimizer of $f(\cdot)$. Furthermore, by selecting different initial iterates for the optimization, or by using different optimization algorithms, one can increase the chance of finding a point that is close to a minimizer of $f(\cdot)$. However, even if the optimization terminates at a point that is non-optimal for $f(\cdot)$, one may have obtained a better system performance compared to not doing any optimization.

See [WP03, WW03] for a further discussion of optimization problems in which the cost function value is computed by a building simulation program.

4.2 Algorithm Selection

In this section, we will discuss which of GenOpt's algorithms can be selected for the optimization problems that we introduced in Section 4.1.

4.2.1 Problem P_c with $n > 1$

To solve $\mathbf{P}_c$ with $n > 1$, the hybrid algorithm (Section 5.5, page 41) or the GPS implementation of the Hooke-Jeeves algorithm (Section 5.2.2, page 23) can be used, possibly with multiple starting points (Section 5.2.3, page 25). If $f(\cdot)$ is once continuously differentiable and has bounded level sets (or if the constraint set $\mathbf{X}$ defined in (4.1) is compact) then these algorithms construct for problem (4.2) a sequence of iterates with stationary accumulation points (see Theorem 5.1.13).

Alternatively, the Discrete Armijo Gradient algorithm (Section 5.3, page 27)

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

can be used. Every accumulation point of the Discrete Armijo Gradient algorithm is a feasible stationary point.

If $f(\cdot)$ is not continuously differentiable, or if $f(\cdot)$ must be approximated by an approximating cost function $f^*(\epsilon, \cdot)$ where the approximation error cannot be controlled, as described in Section 4.1.4, then $\mathbf{P}_c$ can only be solved heuristically. We recommend using the hybrid algorithm (Section 5.5, page 41), the GPS implementation of the Hooke-Jeeves algorithm (Section 5.2.2, page 23), possibly with multiple starting points (Section 5.2.3, page 25), or a Particle Swarm Optimization algorithm (Section 5.4, page 31).

We do not recommend using the Nelder-Mead Simplex algorithm (Section 5.6, page 44) or the Discrete Armijo Gradient algorithm (Section 5.3, page 27).

The following approach reduces the risk of failing at a point which is non-optimal and far from a minimizer of $f(\cdot)$:

1. Selecting large values for the parameter `Step` in the optimization command file (see page 84).
2. Selecting different initial iterates.
3. Using the hybrid algorithm of Section 5.5, the GPS implementation of the Hooke-Jeeves algorithm, possibly with multiple starting points (Section 5.2.3, page 25), and/or a Particle Swarm Optimization algorithm and select the best of the solutions.
4. Doing a parametric study around the solution that has been obtained by any of the above optimization algorithms. The parametric study can be done using the algorithms `Parametric` (Section 7.1, page 61) and/or `EquMesh` (Section 7.2, page 62). If the parametric study yields a further reduction in cost, then the optimization failed at a non-optimal point. In this situation, one may want to try another optimization algorithm.

If $f(\cdot)$ is continuously differentiable but must be approximated by approximating cost functions $f^*(\epsilon, \cdot)$ where the approximation error can be controlled as described in Section 4.1.4, then $\mathbf{P}_c$ can be solved using the hybrid algorithm (Section 5.5, page 41) or the GPS implementation of the Hooke-Jeeves algorithm (Section 5.2.2, page 23), both with the error control scheme described in the Model GPS Algorithm 5.1.8 (page 18). The GPS implementation of the Hooke-Jeeves algorithm can be used with multiple starting points (Section 5.2.3, page 25). The error control scheme can be implemented using the value of GenOpt's variable `stepNumber` (page 67) and GenOpt's pre-processing capabilities (Section 11.3, page 89). A more detailed description of how to use the error control scheme can be found in [PW03, WP03].

4.2.2 Problem P_{cg} with $n > 1$

To solve $\mathbf{P}_{cg}$, the hybrid algorithm (Section 5.5, page 41) or the GPS implementation of the Hooke-Jeeves algorithm (Section 5.2.2, page 23) can be used,

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

possibly with multiple starting points (Section 5.2.3, page 25). Constraints $g(\cdot) \leq 0$ can be implemented using barrier and penalty functions (Section 8, page 64).

If $f(\cdot)$ or $g(\cdot)$ are not continuously differentiable, we recommend using the hybrid algorithm (Section 5.5, page 41) or the GPS implementation of the Hooke-Jeeves algorithm (Section 5.2.2, page 23), possibly with multiple starting points (Section 5.2.3, page 25), and implement the constraints $g(\cdot) \leq 0$ using barrier and penalty functions (Section 8, page 64). To reduce the risk of terminating far from a minimum point of $f(\cdot)$, we recommend the same measures as for solving $\mathbf{P}_c$.

4.2.3 Problem P_c with $n = 1$

To solve $\mathbf{P}_c$ with $n = 1$, any of the interval division algorithms can be used (Section 6.1, page 55). Since only a few function evaluations are required for parametric studies in one dimension, the algorithm `Parametric` can also be used for this problem (Section 7.1, page 61). We recommend doing a parametric study if $f(\cdot)$ is expected to have several local minima.

4.2.4 Problem P_{cg} with $n = 1$

To solve $\mathbf{P}_{cg}$ with $n = 1$, the same applies as for $\mathbf{P}_c$ with $n = 1$. Constraints $g(\cdot) \leq 0$ can be implemented by setting the penalty weighting factor μ in (8.8) to a large value. This may still cause small constraint violations, but it is easy to check whether the violation is acceptable.

4.2.5 Problem P_d

To solve $\mathbf{P}_d$, a Particle Swarm Optimization algorithm can be used (Section 5.4, page 31).

4.2.6 Problem P_{cd} and P_{cdg}

To solve $\mathbf{P}_{cd}$, or $\mathbf{P}_{cdg}$, the hybrid algorithm (Section 5.5, page 41) or a Particle Swarm Optimization algorithm can be used (Section 5.4, page 31).

4.2.7 Functions with Several Local Minima

If the problem has several local minima, we recommend using the GPS implementation of the Hooke-Jeeves algorithm with multiple starting points (Section 5.2.3, page 25), the hybrid algorithm (Section 5.5, page 41), or a Particle Swarm Optimization algorithm (Section 5.4, page 31).

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

5 Algorithms for Multi-Dimensional Optimization

5.1 Generalized Pattern Search Methods (Analysis)

Generalized Pattern Search (GPS) algorithms are derivative free optimization algorithms for the minimization of problem $\mathbf{P}_c$ and $\mathbf{P}_{cg}$, defined in (4.2) and (4.3), respectively. We will present the GPS algorithms for the case where the function $f(\cdot)$ cannot be evaluated exactly, but can be approximated by functions $f^*\colon \mathbb{R}^q_+ \times \mathbb{R}^n \to \mathbb{R}$, where the first argument $\epsilon \in \mathbb{R}^q_+$ is the precision parameter of PDE, ODE, and algebraic equation solvers. Obviously, the explanations are similar for problems where $f(\cdot)$ can be evaluated exactly, except that the scheme to control ϵ is not applicable, and that the approximate functions $f^*(\epsilon, \cdot)$ are replaced by $f(\cdot)$.

Under the assumption that the cost function is continuously differentiable, all the accumulation points constructed by the GPS algorithms are stationary.

What GPS algorithms have in common is that they define the construction of a mesh $\mathbb{M}_k$ in $\mathbb{R}^n$, which is then explored according to some rules that differ among the various members of the family of GPS algorithms. If no decrease in cost is obtained on mesh points around the current iterate, then the distance between the mesh points is reduced, and the process is repeated.

We will now explain the framework of GPS algorithms that will be used to implement different instances of GPS algorithms in GenOpt. The discussion follows the more detailed description of [PW03].

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

5.1.1 Assumptions

We will assume that $f(\cdot)$ and its approximating functions $\{f^*(\epsilon,\cdot)\}_{\epsilon\in\mathbb{R}^q_+}$ have the following properties.

Assumption 5.1.1

1. *There exists an error bound function $\varphi\colon \mathbb{R}^q_+ \to \mathbb{R}_+$ such that for any bounded set $\mathbf{S}\subset\mathbf{X}$, there exists an $\epsilon_{\mathbf{S}}\in\mathbb{R}^q_+$ and a scalar $K_{\mathbf{S}}\in(0,\infty)$ such that for all $x\in\mathbf{S}$ and for all $\epsilon\in\mathbb{R}^q_+$, with $\epsilon\le\epsilon_{\mathbf{S}}$,*[1]

$$|\, f^*(\epsilon,x)-f(x)| \le K_{\mathbf{S}}\,\varphi(\epsilon). \tag{5.1}$$

   Furthermore,

$$\lim_{\|\epsilon\|\to 0}\varphi(\epsilon)=0. \tag{5.2}$$

2. *The function $f\colon\mathbb{R}^n\to\mathbb{R}$ is once continuously differentiable.* □

Remark 5.1.2

1. The functions $\{f^*(\epsilon,\cdot)\}_{\epsilon\in\mathbb{R}^q_+}$ may be discontinuous.
2. See [PW03] for the situation where $f(\cdot)$ is only locally Lipschitz continuous. □

Next, we state an assumption on the level sets of the family of approximate functions. To do so, we first define the notion of a level set.

Definition 5.1.3 (Level Set) *Given a function $f\colon\mathbb{R}^n\to\mathbb{R}$ and an $\alpha\in\mathbb{R}$, such that $\alpha>\inf_{x\in\mathbb{R}^n} f(x)$, we will say that the set $\mathbf{L}_\alpha(f)\subset\mathbb{R}^n$, defined as*

$$\mathbf{L}_\alpha(f)\triangleq\{x\in\mathbb{R}^n \mid f(x)\le\alpha\}, \tag{5.3}$$

is a level set *of $f(\cdot)$, parametrized by α.* □

Assumption 5.1.4 (Compactness of Level Sets) *Let $\{f^*(\epsilon,\cdot)\}_{\epsilon\in\mathbb{R}^q_+}$ be as in Assumption 5.1.1 and let $\mathbf{X}\subset\mathbb{R}^n$ be the constraint set. Let $x_0\in\mathbf{X}$ be the initial iterate and $\epsilon_0\in\mathbb{R}^q_+$ be the initial precision setting of the numerical solvers. Then, we assume that there exists a compact set $\mathbf{C}\subset\mathbb{R}^n$ such that*

$$\mathbf{L}_{f^*(\epsilon_0,x_0)}(f^*(\epsilon,\cdot))\cap\mathbf{X}\subset\mathbf{C},\qquad\forall\,\epsilon\le\epsilon_0. \tag{5.4}$$

□

[1] For $\epsilon\in\mathbb{R}^q_+$, by $\epsilon\le\epsilon_{\mathbf{S}}$, we mean that $0<\epsilon^i\le\epsilon^i_{\mathbf{S}}$, for all $i\in\{1,\ldots,q\}$.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

5.1.2 Characterization of GPS Algorithms

There exist different geometrical explanations for pattern search algorithms, and a generalization is given in the review [KLT03]. We will use a simple implementation of the pattern search algorithms in [PW03] where we restrict the search directions to be the positive and negative coordinate directions. Thus, the search directions are the columns of the matrix

$$D \triangleq [-e_1,\, +e_1,\, \ldots\, ,-e_n,\, +e_n] \in \mathbb{Z}^{n\times 2n}, \tag{5.5}$$

which suffices for box-constrained problems. Furthermore, we construct the sequence of mesh size parameters that parametrizes the minimum distance between iterates such that it satisfies the following assumption.

Assumption 5.1.5 (k-th Mesh Size Parameter) *Let $r, s_0, k \in \mathbb{N}$, with $r > 1$, and $\{t_i\}_{i=0}^{k-1} \subset \mathbb{N}$. We will assume that the sequence of mesh size parameters satisfies*

$$\Delta_k \triangleq \frac{1}{r^{s_k}}, \tag{5.6a}$$

where for $k > 0$

$$s_k \triangleq s_0 + \sum_{i=0}^{k-1} t_i. \tag{5.6b}$$

□

With this construction, all iterates lie on a rational mesh of the form

$$\mathbb{M}_k \triangleq \{x_0 + \Delta_k\, D\, m \mid m \in \mathbb{N}^{2n}\}. \tag{5.7}$$

We will now characterize the set-valued maps that determine the mesh points for the "global" and "local" searches. Note that the images of these maps may depend on the entire history of the computation.

Definition 5.1.6 *Let $\underline{\mathbf{X}}_k \subset \mathbb{R}^n$ and $\underline{\mathbf{\Delta}}_k \subset \mathbb{Q}_+$ be the sets of all sequences containing $k+1$ elements, let $\mathbb{M}_k$ be the current mesh, and let $\epsilon \in \mathbb{R}^q_+$ be the solver tolerance.*

1. *We define the* global search set map *to be any set-valued map*

$$\gamma_k \colon \underline{\mathbf{X}}_k \times \underline{\mathbf{\Delta}}_k \times \mathbb{R}^q_+ \to \left(2^{\mathbb{M}_k} \cap \mathbf{X}\right) \cup \emptyset \tag{5.8a}$$

   whose image $\gamma_k(\underline{x}_k, \underline{\Delta}_k, \epsilon)$ contains only a finite number of mesh points.

2. *We will call $\mathcal{G}_k \triangleq \gamma_k(\underline{x}_k, \underline{\Delta}_k, \epsilon)$ the* global search set*.*

3. *We define the directions for the local search as*

$$D \triangleq [-e_1,\, +e_1,\, \ldots\, ,-e_n,\, +e_n]. \tag{5.8b}$$

4. *We will call*

$$\mathcal{L}_k \triangleq \left\{x_k + \Delta_k\, D\, e_i \mid i \in \{1, \ldots, 2\,n\}\right\} \cap \mathbf{X} \tag{5.8c}$$

   the local search set*.*

□

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Remark 5.1.7

1. The map $\gamma_k(\cdot,\cdot,\cdot)$ can be dynamic in the sense that if $\{x_{k_i}\}_{i=0}^{I} \triangleq \gamma_k(\underline{x}_k, \underline{\Delta}_k, \epsilon)$, then the rule for selecting $x_{k_{\widehat{i}}}$, $1 \le \widehat{i} \le I$, can depend on $\{x_{k_i}\}_{i=0}^{\widehat{i}-1}$ and $\{f^*(\epsilon, x_{k_i})\}_{i=0}^{\widehat{i}-1}$. It is only important that the global search terminates after a finite number of computations, and that $\mathcal{G}_k \subset (2^{\mathbb{M}_k} \cap \mathbf{X}) \cup \emptyset$.
2. As we shall see, the global search affects only the efficiency of the algorithm but not its convergence properties. Any heuristic procedure that leads to a finite number of function evaluations can be used for $\gamma_k(\cdot,\cdot,\cdot)$.
3. The empty set is included in the range of $\gamma_k(\cdot,\cdot,\cdot)$ to allow omitting the global search.

□

5.1.3 Model Adaptive Precision GPS Algorithm

We will now present our model GPS algorithm with adaptive precision cost function evaluations.

Algorithm 5.1.8 (Model GPS Algorithm)

Data:	Initial iterate $x_0 \in \mathbf{X}$; Mesh size divider $r \in \mathbb{N}$, with $r > 1$; Initial mesh size exponent $s_0 \in \mathbb{N}$.
Maps:	Global search set map $\gamma_k \colon \underline{\mathbf{X}_k} \times \underline{\mathbf{\Delta}}_k \times \mathbb{R}_+^q \to \left(2^{\mathbb{M}_k} \cap \mathbf{X}\right) \cup \emptyset$; Function $\rho \colon \mathbb{R}_+ \to \mathbb{R}_+^q$ (to assign ϵ), such that the composition $\varphi \circ \rho \colon \mathbb{R}_+ \to \mathbb{R}_+$ is strictly monotone decreasing and satisfies $\varphi(\rho(\Delta))/\Delta \to 0$, as $\Delta \to 0$.
Step 0:	Initialize $k = 0$, $\Delta_0 = 1/r^{s_0}$, and $\epsilon = \rho(1)$.
Step 1:	<u>Global Search</u> Construct the global search set $\mathcal{G}_k = \gamma_k(\underline{x}_k, \underline{\Delta}_k, \epsilon)$. If $f^*(\epsilon, x') - f^*(\epsilon, x_k) < 0$ for any $x' \in \mathcal{G}_k$, go to Step 3; else, go to Step 2.
Step 2:	<u>Local Search</u> Evaluate $f^*(\epsilon, \cdot)$ for any $x' \in \mathcal{L}_k$ until some $x' \in \mathcal{L}_k$ satisfying $f^*(\epsilon, x') - f^*(\epsilon, x_k) < 0$ is obtained, or until all points in $\mathcal{L}_k$ are evaluated.
Step 3:	<u>Parameter Update</u> If there exists an $x' \in \mathcal{G}_k \cup \mathcal{L}_k$ satisfying $f^*(\epsilon, x') - f^*(\epsilon, x_k) < 0$, set $x_{k+1} = x'$, $s_{k+1} = s_k$, $\Delta_{k+1} = \Delta_k$, and do not change ϵ; else, set $x_{k+1} = x_k$, $s_{k+1} = s_k + t_k$, with $t_k \in \mathbb{N}_+$ arbitrary, $\Delta_{k+1} = 1/r^{s_{k+1}}$, $\epsilon = \rho(\Delta_{k+1}/\Delta_0)$.
Step 4:	Replace k by $k+1$, and go to Step 1.

Remark 5.1.9

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

1. To ensure that ϵ does not depend on the scaling of Δ_0, we normalized the argument of $\rho(\cdot)$. In particular, we want to decouple ϵ from the user's choice of the initial mesh parameter.

2. In Step 2, once a decrease of the cost function is obtained, one can proceed to Step 3. However, one is allowed to evaluate $f^*(\epsilon, \cdot)$ at more points in $\mathcal{L}_k$ in an attempt to obtain a bigger reduction in cost. However, one is allowed to proceed to Step 3 only after either a cost decrease has been found, or after *all* points in $\mathcal{L}_k$ are tested.

3. In Step 3, we are not restricted to accepting the $x' \in \mathcal{G}_k \cup \mathcal{L}_k$ that gives lowest cost value. But the mesh size parameter Δ_k is reduced *only* if there exists no $x' \in \mathcal{G}_k \cup \mathcal{L}_k$ satisfying $f^*(\epsilon, x') - f^*(\epsilon, x_k) < 0$.

4. To simplify the explanations, we do not increase the mesh size parameter if the cost has been reduced. However, our global search allows searching on a coarser mesh $\widehat{\mathbb{M}} \subset \mathbb{M}_k$, and hence, our algorithm can easily be extended to include a rule for increasing Δ_k for a finite number of iterations.

5. Audet and Dennis [AD03] update the mesh size parameter using the formula $\Delta_{k+1} = \tau^m \Delta_k$, where $\tau \in \mathbb{Q}$, $\tau > 1$, and m is any element of $\mathbb{Z}$. Thus, our update rule for Δ_k is a special case of Audet's and Dennis' construction since we set $\tau = 1/r$, with $r \in \mathbb{N}_+$, $r \geq 2$ (so that $\tau < 1$) and $m \in \mathbb{N}$. We prefer our construction because we do not think it negatively affects the computing performance, but it leads to simpler convergence proofs. □

5.1.4 Convergence Results

We will now present the convergence results for our Model GPS algorithm. See [PW03] for a detailed discussion and convergence proofs.

a) Unconstrained Minimization

We will first present the convergence properties of the Model GPS Algorithm 5.1.8 on unconstrained minimization problems, i.e., for $\mathbf{X} = \mathbb{R}^n$.

First, we will need the notion of a *refining subsequence*, which we define as follows:

Definition 5.1.10 (Refining Subsequence) *Consider a sequence $\{x_k\}_{k=0}^{\infty}$ constructed by Model GPS Algorithm 5.1.8. We will say that the subsequence $\{x_k\}_{k \in \mathbf{K}}$ is the* refining subsequence*, if $\Delta_{k+1} < \Delta_k$ for all $k \in \mathbf{K}$, and $\Delta_{k+1} = \Delta_k$ for all $k \notin \mathbf{K}$.* □

We now state that pattern search algorithms with adaptive precision function evaluations construct sequences with stationary accumulation points.

Theorem 5.1.11 (Convergence to a Stationary Point) *Suppose that Assumptions 5.1.1 and 5.1.4 are satisfied and that $\mathbf{X} = \mathbb{R}^n$. Let $x^* \in \mathbb{R}^n$ be an*

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

accumulation point of the refining subsequence $\{x_k\}_{k\in\mathbf{K}}$*, constructed by Model GPS Algorithm 5.1.8. Then,*

$$\nabla f(x^*) = 0. \tag{5.9}$$

□

b) Box-Constrained Minimization

We now present the convergence results for the box-constrained problem (4.2). See [AD03, PW03, KLT03] for the more general case of linearly-constrained problems and for the convergence proofs.

First, we introduce the notion of a tangent cone and a normal cone, which are defined as follows:

Definition 5.1.12 (Tangent and Normal Cone)

1. *Let* $\mathbf{X} \subset \mathbb{R}^n$*. Then, we define the* tangent cone *to* $\mathbf{X}$ *at a point* $x^* \in \mathbf{X}$ *by*
$$\mathbf{T}_{\mathbf{X}}(x^*) \triangleq \overline{\{\mu\,(x - x^*) \mid \mu \geq 0,\, x \in \mathbf{X}\}}. \tag{5.10a}$$
2. *Let* $\mathbf{T}_{\mathbf{X}}(x^*)$ *be as above. Then, we define the* normal cone *to* $\mathbf{X}$ *at* $x^* \in \mathbf{X}$ *by*
$$\mathbf{N}_{\mathbf{X}}(x^*) \triangleq \{v \in \mathbb{R}^n \mid \forall\, t \in \mathbf{T}_{\mathbf{X}}(x^*),\, \langle v,\, t\rangle \leq 0\}. \tag{5.10b}$$

□

We now state that the accumulation points generated by Model GPS Algorithm 5.1.8 are feasible stationary points of problem (4.2).

Theorem 5.1.13 (Convergence to a Feasible Stationary Point)
Suppose Assumptions 5.1.1 and 5.1.4 are satisfied. Let $x^* \in \mathbf{X}$ *be an accumulation point of a refining subsequence* $\{x_k\}_{k\in\mathbf{K}}$ *constructed by Model GPS Algorithm 5.1.8 in solving problem* (4.2)*. Then,*

$$\langle \nabla f(x^*),\, t\rangle \geq 0, \qquad \forall\, t \in \mathbf{T}_{\mathbf{X}}(x^*), \tag{5.11a}$$

and

$$-\nabla f(x^*) \in \mathbf{N}_{\mathbf{X}}(x^*). \tag{5.11b}$$

5.2 Generalized Pattern Search Methods (Implementations)

We will now present different implementations of the Generalized Pattern Search (GPS) algorithms. They all use the Model GPS Algorithm 5.1.8 to solve problem $\mathbf{P}_c$ defined in (4.2). The problem $\mathbf{P}_{cg}$ defined in (4.3) can be solved by using penalty functions as described in Section 8.2.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

We will discuss the implementations for the case where the function $f(\cdot)$ cannot be evaluated exactly, but will be approximated by functions $f^*\colon \mathbb{R}^q_+ \times \mathbb{R}^n \to \mathbb{R}$, where the first argument $\epsilon \in \mathbb{R}^q_+$ is the precision parameter of the PDE, ODE, and algebraic equation solvers. This includes the case where ϵ is not varied during the optimization, in which case the explanations are identical, except that the scheme to control ϵ is not applicable, and that the approximate functions $f^*(\epsilon, \cdot)$ are replaced by $f(\cdot)$.

If the cost function $f(\cdot)$ is approximated by functions $\{f^*(\epsilon, \cdot)\}_{\epsilon \in \mathbb{R}^q_+}$ with adaptive precision ϵ, then the function $\rho\colon \mathbb{R}_+ \to \mathbb{R}^q_+$ (to assign ϵ) can be implemented by using GenOpt's pre-processing capability (see Section 11.3).

5.2.1 Coordinate Search Algorithm

We will now present the implementation of the Coordinate Search algorithm with adaptive precision function evaluations using the Model GPS Algorithm 5.1.8. To simplify the implementation, we assign $f^*(\epsilon, x) = \infty$ for all $x \notin \mathbf{X}$ where $\mathbf{X}$ is defined in (4.1).

a) Algorithm Parameters

The search direction matrix is defined as

$$D \triangleq [+s^1\, e_1,\, -s^1\, e_1, \ldots,\, +s^n\, e_n,\, -s^n\, e_n] \tag{5.12}$$

where $s^i \in \mathbb{R}$, $i \in \{1, \ldots, n\}$, is a scaling for each parameter (specified by GenOpt's parameter `Step`).

The parameter $r \in \mathbb{N}$, $r > 1$, which is used to compute the mesh size parameter Δ_k, is defined by the parameter `MeshSizeDivider`, the initial value for the mesh size exponent $s_0 \in \mathbb{N}$ is defined by the parameter `InitialMeshSizeExponent`, and the mesh size exponent increment t_k is, for the iterations that do not reduce the cost, defined by the parameter `MeshSizeExponentIncrement`.

b) Global Search

In the Coordinate Search Algorithm, there is no global search. Thus, $\mathcal{G}_k = \emptyset$ for all $k \in \mathbb{N}$.

c) Local Search

The local search set $\mathcal{G}_k$ is constructed using the set-valued map $E_k\colon \mathbb{R}^n \times \mathbb{Q}_+ \times \mathbb{R}^q_+ \to 2^{\mathbb{M}_k}$, which is defined as follows:

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Algorithm 5.2.1 (Map $E_k \colon \mathbb{R}^n \times \mathbb{Q}_+ \times \mathbb{R}^q_+ \to 2^{\mathbb{M}_k}$ for "Coordinate Search")

Parameter:	Search direction matrix $D = [+s^1\, e_1,\, -s^1\, e_1, \ldots,\, +s^n\, e_n,\, -s^n\, e_n]$.
	Vector $\mu \in \mathbb{N}^n$.
Input:	Iteration number $k \in \mathbb{N}$.
	Base point $x \in \mathbb{R}^n$.
	Mesh divider $\Delta_k \in \mathbb{Q}_+$.
Output:	Set of trial points $\mathcal{T}$.
Step 0:	Initialize $\mathcal{T} = \emptyset$.
	If $k = 0$, initialize , $\mu^i = 0$ for all $i \in \{1, \ldots, n\}$.
Step 1:	For $i = 1, \ldots, n$
	Set $\widetilde{x} = x + \Delta_k\, D\, e_{2\,i-1+\mu^i}$ and $\mathcal{T} \leftarrow \mathcal{T} \cup \{\widetilde{x}\}$.
	If $f^*(\epsilon, \widetilde{x}) < f^*(\epsilon, x)$
	Set $x = \widetilde{x}$.
	else
	If $\mu^i = 0$, set $\mu^i = 1$, else set $\mu^i = 0$.
	Set $\widetilde{x} = x + \Delta_k\, D\, e_{2\,i-1+\mu_i}$ and $\mathcal{T} \leftarrow \mathcal{T} \cup \{\widetilde{x}\}$.
	If $f^*(\epsilon, \widetilde{x}) < f^*(\epsilon, x)$
	Set $x = \widetilde{x}$.
	else
	If $\mu^i = 0$, set $\mu^i = 1$, else set $\mu^i = 0$.
	end if.
	end if.
	end for.
Step 2:	Return $\mathcal{T}$.

Thus, $E_k(x, \Delta_k, \epsilon) = \mathcal{T}$ for all $k \in \mathbb{N}$.

Remark 5.2.2 In Algorithm 5.2.1, the vector $\mu \in \mathbb{N}^n$ contains for each coordinate direction an integer 0 or 1 that indicates whether a step in the positive or in the negative coordinate direction yield a decrease in cost in the previous iteration. This reduces the number of exploration steps. □

d) Parameter Update

The point x' in Step 3 of the GPS Model Algorithm 5.1.8 corresponds to $x' \triangleq \arg\min_{x \in E_k(x_k, \Delta_k, \epsilon)} f^*(\epsilon, x)$ in the Coordinate Search algorithm.

e) Keywords

For the GPS implementation of the Coordinate Search Algorithm, the command file (see page 83) must only contain continuous parameters.

To invoke the algorithm, the `Algorithm` section of the GenOpt command file must have the following form:

```
Algorithm{
   Main                        = GPSCoordinateSearch;
```

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

```
   MeshSizeDivider           = Integer;   // 1 <  MeshSizeDivider
   InitialMeshSizeExponent   = Integer;   // 0 <= InitialMeshSizeExponent
   MeshSizeExponentIncrement = Integer;   // 0 <  MeshSizeExponentIncrement
   NumberOfStepReduction     = Integer;   // 0 <  NumberOfStepReduction
}
```

The entries are defined as follows:

`Main` The name of the main algorithm.

`MeshSizeDivider` The value for $r \in \mathbb{N}$, $r > 1$, used to compute $\Delta_k \triangleq 1/r^{s_k}$ (see equation (5.6a)). A common value is $r = 2$.

`InitialMeshSizeExponent` The value for $s_0 \in \mathbb{N}$ in (5.6b). A common value is $s_0 = 0$.

`MeshSizeExponentIncrement` The value for $t_i \in \mathbb{N}$ (for the iterations that do not yield a decrease in cost) in (5.6b). A common value is $t_i = 1$.

`NumberOfStepReduction` The maximum number of step reductions before the algorithm stops. Thus, if we use the notation $m \triangleq$ `NumberOfStepReduction`, then we have for the last iterations $\Delta_k = 1/r^{s_0 + m\, t_k}$. A common value is $m = 4$.

5.2.2 Hooke-Jeeves Algorithm

We will now present the implementation of the Hooke-Jeeves algorithm [HJ61] with adaptive precision function evaluations using the Model GPS Algorithm 5.1.8. The modifications of Smith [Smi69], Bell and Pike [BP66] and De Vogelaere [DV68] are implemented in this algorithm.

To simplify the implementation, we assign $f^*(\epsilon, x) = \infty$ for all $x \notin \mathbf{X}$ where $\mathbf{X}$ is defined in (4.1).

a) Algorithm Parameters

The algorithm parameters D, r, s_0, and t_k are defined as in the Coordinate Search algorithm (see page 21).

b) Map for Exploratory Moves

To facilitate the algorithm explanation, we use the set-valued map $E_k : \mathbb{R}^n \times \mathbb{Q}_+ \times \mathbb{R}^q_+ \to 2^{\mathbb{M}_k}$, as defined in Algorithm 5.2.1. The map $E_k(\cdot,\cdot,\cdot)$ defines the "exploratory moves" in [HJ61], and will be used in Section c) to define the global search set map and, under conditions to be seen in Section d), the local search direction map as well.

c) Global Search Set Map

The global search set map $\gamma_k(\cdot,\cdot,\cdot)$ is defined as follows. Because $\gamma_0(\cdot,\cdot,\cdot)$ depends on x_{-1}, we need to introduce x_{-1}, which we define as $x_{-1} \triangleq x_0$.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Algorithm 5.2.3 (Global Search Set Map $\gamma_k \colon \underline{\mathbf{X}}_k \times \underline{\mathbf{\Delta}}_k \times \mathbb{R}^q_+ \to 2^{\mathbb{M}_k}$)

Map:	Map for "exploratory moves" $E_k \colon \mathbb{R}^n \times \mathbb{Q}_+ \times \mathbb{R}^q_+ \to 2^{\mathbb{M}_k}$.
Input:	Previous and current iterate, $x_{k-1} \in \mathbb{R}^n$ and $x_k \in \mathbb{R}^n$.
	Mesh divider $\Delta_k \in \mathbb{Q}_+$.
	Solver precision $\epsilon \in \mathbb{R}^q_+$.
Output:	Global search set $\mathcal{G}_k$.
Step 1:	Set $x = x_k + (x_k - x_{k-1})$.
Step 2:	Compute $\mathcal{G}_k = E_k(x, \Delta_k, \epsilon)$.
Step 3:	If $\left(\min_{x \in \mathcal{G}_k} f^*(\epsilon, x)\right) > f^*(\epsilon, x_k)$
	Set $\mathcal{G}_k \leftarrow \mathcal{G}_k \cup E_k(x_k, \Delta_k, \epsilon)$.
	end if.
Step 4:	Return $\mathcal{G}_k$.

Thus, $\gamma_k(\underline{x}_k, \underline{\Delta}_k, \epsilon) = \mathcal{G}_k$.

d) Local Search Direction Map

If the global search, as defined by Algorithm 5.2.3, has failed in reducing $f^*(\epsilon, \cdot)$, then Algorithm 5.2.3 has constructed a set $\mathcal{G}_k$ that contains the set $\{x_k + \Delta_k D e_i \mid i = 1, \ldots, 2n\}$. This is because in the evaluation of $E_k(x_k, \Delta_k, \epsilon)$, defined in Algorithm 5.2.1, all "If $f^*(\epsilon, \widetilde{x}) < f^*(\epsilon, x)$" statements yield `false`, and, hence, one has constructed $\{x_k + \Delta_k D e_i \mid i = 1, \ldots, 2n\} = E_k(x_k, \Delta_k, \epsilon)$.

Because the columns of D span $\mathbb{R}^n$ positively, it follows that the search on the set $\{x_k + \Delta_k D e_i \mid i = 1, \ldots, 2n\}$ is a local search. Hence, the constructed set

$$\mathcal{L}_k \triangleq \{x_k + \Delta_k D e_i \mid i = 1, \ldots, 2n\} \subset \mathcal{G}_k \tag{5.13}$$

is a local search set. Consequently, $f^*(\epsilon, \cdot)$ has already been evaluated at all points of $\mathcal{L}_k$ (during the construction of $\mathcal{G}_k$) and, hence, one does not need to evaluate $f^*(\epsilon, \cdot)$ again in a local search.

e) Parameter Update

The point x' in Step 3 of the GPS Model Algorithm 5.1.8 corresponds to $x' \triangleq \arg\min_{x \in \mathcal{G}_k} f^*(\epsilon, x)$ in the Hooke-Jeeves algorithm. (Note that $\mathcal{L}_k \subset \mathcal{G}_k$ if a local search has been done as explained in the above paragraph.)

f) Keywords

For the GPS implementation of the Hooke-Jeeves algorithm, the command file (see page 83) must only contain continuous parameters.

To invoke the algorithm, the `Algorithm` section of the GenOpt command file must have the following form:

```
Algorithm{
   Main                        = GPSHookeJeeves;
```

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

```
   MeshSizeDivider           = Integer;   // bigger than 1
   InitialMeshSizeExponent   = Integer;   // bigger than or equal to 0
   MeshSizeExponentIncrement = Integer;   // bigger than 0
   NumberOfStepReduction     = Integer;   // bigger than 0
}
```

The entries are the same as for the Coordinate Search algorithm, and explained on page 22.

5.2.3 Multi-Start GPS Algorithms

All GPS algorithms can also be run using multiple initial points. Using multiple initial points increases the chance of finding the global minimum if the cost function has several local minima, and furthermore, it decreases the risk of not finding a minimum if the cost function is not continuously differentiable, which is the case if building simulation programs, such as EnergyPlus or TRNSYS, are used to compute the cost function (see the discussion in Section 4.1.4).

The values that are specified by GenOpt's parameter `Ini` in GenOpt's command file (see Section 11.1.3) are used to initialize the first initial point. The other initial points are randomly distributed, with a uniform distribution, between the lower and upper bounds of the feasible domain. They are, however, set to the mesh $\mathbb{M}_0$, defined in (5.7), which reduces the number of cost function evaluations if the optimization algorithm converges from different initial points to the same minimizer.

In GenOpt's command file, a lower and an upper bound must be specified for each independent variable using the keywords `Min` and `Max`.

To use the `GPSCoordinateSearch` algorithm with multiple starting points, the `Algorithm` section of the GenOpt command file must have the following form:

```
Algorithm{
   Main                      = GPSCoordinateSearch;
   MultiStart                = Uniform;
   Seed                      = Integer;
   NumberOfInitialPoint      = Integer;   // bigger than or equal to 1
   MeshSizeDivider           = Integer;   // 1 <  MeshSizeDivider
   InitialMeshSizeExponent   = Integer;   // 0 <= InitialMeshSizeExponent
   MeshSizeExponentIncrement = Integer;   // 0 <  MeshSizeExponentIncrement
   NumberOfStepReduction     = Integer;   // 0 <  NumberOfStepReduction
}
```

The entries are defined as follows:

`Main` The name of the main algorithm.

`MultiStart` Keyword to invoke the multi-start algorithm. The only valid value is `Uniform`.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

`Seed` This value is used to initialize the random number generator.

`NumberOfInitialPoint` The number of initial points.

The other entries are the same as for the Coordinate Search algorithm, and are explained on page 22.

To use the `GPSHookeJeeves` algorithm with multiple starting points, the `Algorithm` section of the GenOpt command file must have the following form:

```
Algorithm{
   Main                     = GPSHookeJeeves;
   MultiStart               = Uniform;
   Seed                     = Integer;
   NumberOfInitialPoint     = Integer;   // 0 <  NumberOfInitialPoint
   MeshSizeDivider          = Integer;   // 1 <  MeshSizeDivider
   InitialMeshSizeExponent  = Integer;   // 0 <= InitialMeshSizeExponent
   MeshSizeExponentIncrement = Integer;  // 0 <  MeshSizeExponentIncrement
   NumberOfStepReduction    = Integer;   // 0 <  NumberOfStepReduction
}
```

The entries are the same as for the multi-start Coordinate Search algorithm above.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

5.3 Discrete Armijo Gradient

The Discrete Armijo Gradient algorithm can be used to solve problem $\mathbf{P}_c$ defined in (4.2) where $f(\cdot)$ is continuously differentiable.

The Discrete Armijo Gradient algorithm approximates gradients by finite differences. It can be used for problems where the cost function is evaluated by computer code that defines a continuously differentiable function but for which obtaining analytical expressions for the gradients is impractical or impossible.

Since the Discrete Armijo Gradient algorithm is sensitive to discontinuities in the cost function, we recommend not to use this algorithm if the simulation program contains adaptive solvers with loose precision settings, such as EnergyPlus [CLW$^+$01]. On such functions, the algorithm is likely to fail. In Section 4.2, we recommend algorithms that are better suited for such situations.

We will now present the Discrete Armijo Gradient algorithm and the Armijo step-size subprocedure.

Algorithm 5.3.1 (Discrete Armijo Gradient Algorithm)

Data:	Initial iterate $x_0 \in \mathbf{X}$. $\alpha, \beta \in (0,1)$, $\gamma \in (0,\infty)$, $k^*, k_0 \in \mathbb{Z}$, $l_{max}, \kappa \in \mathbb{N}$ (for reseting the step-size calculation). Termination criteria $\epsilon_m, \epsilon_x \in \mathbb{R}_+$, $i_{max} \in \mathbb{N}$.
Step 0:	Initialize $i = 0$ and $m = 0$.
Step 1:	<u>Compute the *search direction* h_i.</u> If $\beta^m < \epsilon_m$, stop. Else, set $\epsilon = \beta^{k_0+m}$ and compute, for $j \in \{1,\ldots,n\}$, $h_i^j = -\left(f(x_i + \epsilon\, e_j) - f(x_i)\right)/\epsilon$.
Step 2 :	<u>Check descent.</u> Compute $\Delta(x_i; h_i) = \left(f(x_i + \epsilon\, h_i) - f(x_i)\right)/\epsilon$. If $\Delta(x_i; h_i) < 0$, go to Step 3. Else, replace m by $m+1$ and go to Step 1.
Step 3 :	<u>Line search.</u> Use Algorithm 5.3.2 (which requires k^*, l_{max} and κ) to compute k_i. Set $$\lambda_i = \underset{\lambda \in \{\beta^{k_i}, \beta^{k_i-1}\}}{\arg\min} f(x_i + \lambda\, h_i). \qquad (5.14)$$
Step 4 :	If $f(x_i + \lambda_i\, h_i) - f(x_i) > -\gamma\, \epsilon$, replace m by $m+1$ and go to Step 1.
Step 5 :	Set $x_{i+1} = x_i + \lambda_i\, h_i$. If $\|\lambda_i\, h_i\| < \epsilon_x$, stop. Else, replace i by $i+1$ and go to Step 1.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Algorithm 5.3.2 (Armijo Step-Size Subprocedure)

Data: Iteration number $i \in \mathbb{N}$, iterate $x_i \in \mathbb{R}^n$, search direction $h_i \in \mathbb{R}^n$, $k^*, k_{i-1} \in \mathbb{Z}$, $\alpha, \beta \in (0,\ 1)$, and $\Delta(x_i; h_i) \in \mathbb{R}$ with $\Delta(x_i; h_i) < 0$, parameter for restart $l_{max}, \kappa \in \mathbb{N}$.

Step 0: Initialize $l = 0$.
If $i = 0$, set $k' = k^*$, else set $k' = k_{i-1}$.

Step 1: Replace l by $l + 1$, and test the conditions

$$f(x_i + \beta^{k'} h_i) - f(x_i) \leq \beta^{k'} \alpha \Delta(x_i; h_i), \quad (5.15a)$$

$$f(x_i + \beta^{k'-1} h_i) - f(x_i) > \beta^{k'-1} \alpha \Delta(x_i; h_i). \quad (5.15b)$$

Step 2: If k' satisfies (5.15a) and (5.15b), return k'.

Step 3: If k' satisfies (5.15b) but not (5.15a),
replace k' by $k' + 1$.
else,
replace k' by $k' - 1$.
If $l < l_{max}$ or $k_{i-1} \leq k^* + \kappa$, go to Step 1. Else, go to Step 4.

Step 4: Set $\mathbf{K} \triangleq \{k \in \mathbb{Z} \mid k \geq k^*\}$, and compute
$k' \triangleq \min_{k \in \mathbf{K}} \{k \mid f(x_i + \beta^k h_i) - f(x_i) \leq \beta^k \alpha \Delta(x_i; h_i)\}$.
Return k'.

Note that in Algorithm 5.3.2, as $\beta \to 1$, the number of tries to compute the Armijo step-size is likely to go to infinity. Under appropriate assumptions one can show that $\alpha = 1/2$ yields fastest convergence [Pol97].

The step-size Algorithm 5.3.2 requires often only a small number of function evaluations. However, occasionally, once a very small step-size has occurred, Algorithm 5.3.2 can trap the Discrete Armijo Gradient algorithm into using a very small step-size for all subsequent iterations. Hence, if $k_{i-1} > k^* + \kappa$, we reset the step-size by computing Step 4.

Algorithm 5.3.1 together with the step-size Algorithm 5.3.2 have the following convergence properties [Pol97].

Theorem 5.3.3 *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable and bounded below.*

1. *If Algorithm 5.3.1 jams at x_i, cycling indefinitely in the loop defined by Steps 1-2 or in the loop defined by Steps 1-4, then $\nabla f(x_i) = 0$.*
2. *If $\{x_i\}_{i=0}^{\infty}$ is an infinite sequence constructed by Algorithm 5.3.1 and Algorithm 5.3.2 in solving* (4.2)*, then every accumulation point $\widehat{x}$ of $\{x_i\}_{i=0}^{\infty}$ satisfies $\nabla f(\widehat{x}) = 0$.*

□

Note that ϵh_i has the same units as the cost function, and the algorithm evaluates $x_i + \lambda h_i$ for some $\lambda \in \mathbb{R}_+$. Thus, the algorithm is sensitive to the

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

scaling of the problem variables, a rather undesirable effect. Therefore, in the implementation of Algorithm 5.3.1 and Algorithm 5.3.2, we normalize the cost function values by replacing, for all $x \in \mathbb{R}^n$, $f(x)$ by $f(x)/f(x_0)$, where x_0 is the initial iterate. Furthermore, we set $x_0 = 0$ and evaluate the cost function for the values $\chi^j + x^j\, s^j$, $j \in \{1, \ldots, n\}$, where $x^j \in \mathbb{R}$ is the j-th component of the design parameter computed in Algorithm 5.3.1 or Algorithm 5.3.2 and $\chi^j \in \mathbb{R}$ and $s^j \in \mathbb{R}$ are the setting of the parameters `Ini` and `Step`, respectively, for the j-th design parameter in the optimization command file (see page 83).

In view of the sensitivity of the Discrete Armijo Gradient algorithm to the scaling of the problem variables and the cost function values, the implementation of penalty and barrier functions may cause numerical problems if the penalty is large compared to the unpenalized cost function value.

If box-constraints for the independent parameters are specified, then the transformations (8.2) are used.

5.3.1 Keywords

For the Discrete Armijo Gradient algorithm, the command file (see page 83) must only contain continuous parameters.

To invoke the algorithm, the `Algorithm` section of the GenOpt command file must have the following form:

```
Algorithm{
   Main  = DiscreteArmijoGradient;
   Alpha = Double;     // 0 < Alpha < 1
   Beta  = Double;     // 0 < Beta  < 1
   Gamma = Double;     // 0 < Gamma
   K0    = Integer;
   KStar = Integer;
   LMax  = Integer;    // 0 <= LMax
   Kappa = Integer;    // 0 <= LMax
   EpsilonM = Double;  // 0 < EpsilonM
   EpsilonX = Double;  // 0 < EpsilonX
}
```

The entries are defined as follows:

`Main` The name of the main algorithm.

`Alpha` The variable α used in Step 1 and in Step 4 of Algorithm 5.3.2. A typical value is $\alpha = 1/2$.

`Beta` The variable β used in approximating the gradient and doing the line search. A typical value is $\beta = 0.8$.

`Gamma` The variable γ used in Step 4 of Algorithm 5.3.1 to determine whether the accuracy of the gradient approximation will be increased.

`K0` The variable k_0 that determines the initial accuracy of the gradient approximation.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

`KStar` The variable k^* used to initialize the line search.

`LMax` The variable l_{max} used in Step 3 of Algorithm 5.3.2 to determine whether the line search needs to be reinitialized.

`Kappa` The variable κ used in Step 3 of Algorithm 5.3.2 to determine whether the line search needs to be reinitialized.

`EpsilonM` The variable ϵ_m used in the determination criteria $\beta^m < \epsilon_m$ in Step 1 of Algorithm 5.3.1.

`EpsilonX` The variable ϵ_x used in the determination criteria $\|\lambda_i\, h_i\| < \epsilon_x$ in Step 5 of Algorithm 5.3.1.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

5.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) algorithms are population-based probabilistic optimization algorithms first proposed by Kennedy and Eberhart [EK95, KE95] to solve problem $\mathbf{P}_c$ defined in (4.2) with possibly discontinuous cost function $f\colon \mathbb{R}^n \to \mathbb{R}$. In Section 5.4.2, we will present a PSO algorithm for discrete independent variables to solve problem $\mathbf{P}_d$ defined in (4.4), and in Section 5.4.3 we will present a PSO algorithm for continuous and discrete independent variables to solve problem $\mathbf{P}_{cd}$ defined in (4.6). To avoid ambiguous notation, we always denote the dimension of the continuous independent variable by $n_c \in \mathbb{N}$ and the dimension of the discrete independent variable by $n_d \in \mathbb{N}$.

PSO algorithms exploit a set of potential solutions to the optimization problem. Each potential solution is called a *particle*, and the set of potential solutions in each iteration step is called a *population*. PSO algorithms are global optimization algorithms and do not require nor approximate gradients of the cost function. The first population is typically initialized using a random number generator to spread the particles uniformly in a user-defined hypercube. A particle update equation, which is modeled on the social behavior of members of bird flocks or fish schools, determines the location of each particle in the next generation.

A survey of PSO algorithms can be found in Eberhart and Shi [ES01]. Laskari et. al. present a PSO algorithm for minimax problems [LPV02b] and for integer programming [LPV02a]. In [PV02a], Parsopoulos and Vrahatis discuss the implementation of inequality and equality constraints to solve problem $\mathbf{P}_{cg}$ defined in (4.3).

We first discuss the case where the independent variable is continuous, i.e., the case of problem $\mathbf{P}_c$ defined in (4.2).

5.4.1 PSO for Continuous Variables

We will first present the initial version of the PSO algorithm which is the easiest to understand.

In the initial version of the PSO algorithm [EK95, KE95], the update equation for the particle location is as follows: Let $k \in \mathbb{N}$ denote the generation number, let $n_P \in \mathbb{N}$ denote the number of particles in each generation, let $x_i(k) \in \mathbb{R}^{n_c}$, $i \in \{1, \ldots, n_P\}$, denote the i-th particle of the k-th generation, let $v_i(k) \in \mathbb{R}^{n_c}$ denote its velocity, let $c_1, c_2 \in \mathbb{R}_+$ and let $\rho_1(k), \rho_2(k) \sim U(0,1)$ be uniformly distributed random numbers between 0 and 1. Then, the update equation is, for all $i \in \{1, \ldots, n_P\}$ and all $k \in \mathbb{N}$,

$$\begin{aligned} v_i(k+1) &= v_i(k) + c_1\, \rho_1(k) \big(p_{l,i}(k) - x_i(k)\big) \\ &\quad + c_2\, \rho_2(k) \big(p_{g,i}(k) - x_i(k)\big), && (5.16a) \\ x_i(k+1) &= x_i(k) + v_i(k+1), && (5.16b) \end{aligned}$$

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

where $v_i(0) \triangleq 0$ and

$$p_{l,i}(k) \triangleq \underset{x \in \{x_i(j)\}_{j=0}^{k}}{\arg\min} f(x), \tag{5.17a}$$

$$p_{g,i}(k) \triangleq \underset{x \in \{\{x_i(j)\}_{j=0}^{k}\}_{i=1}^{n_P}}{\arg\min} f(x). \tag{5.17b}$$

Thus, $p_{l,i}(k)$ is the location that for the i-th particle yields the lowest cost over all generations, and $p_{g,i}(k)$ is the location of the best particle over all generations. The term $c_1\ \rho_1(k)\,(p_{l,i}(k) - x_i(k))$ is associated with cognition since it takes into account the particle's own experience, and the term $c_2\,\rho_2(k)\,(p_{g,i}(k) - x_i(k))$ is associated with social interaction between the particles. In view of this similarity, c_1 is called *cognitive acceleration constant* and c_2 is called *social acceleration constant*.

a) Neighborhood Topology

The minimum in (5.17b) need not be taken over all points in the population. The set of points over which the minimum is taken is defined by the *neighborhood topology*. In PSO, the neighborhood topologies are usually defined using the particle index, and not the particle location. We will use the *lbest*, *gbest*, and the *von Neumann* neighborhood topology, which we will now define.

In the *lbest* topology of size $l \in \mathbb{N}$, with $l > 1$, the neighborhood of a particle with index $i \in \{1, \ldots, n_P\}$ consist of all particles whose index are in the set

$$\mathcal{N}_i \triangleq \{i - l, \ldots i, \ldots, i + l\}, \tag{5.18a}$$

where we assume that the indices wrap around, i.e., we replace -1 by $n_P - 1$, replace -2 by $n_P - 2$, etc.

In the *gbest* topology, the neighborhood contains all points of the population, i.e.,

$$\mathcal{N}_i \triangleq \{1, \ldots, n_P\}, \tag{5.18b}$$

for all $i \in \{1, \ldots, n_P\}$.

For the *von Neumann* topology, consider a 2-dimensional lattice, with the lattice points enumerated as shown in Figure 5.1. We will use the von Neumann topology of range 1, which is defined, for $i, j \in \mathbb{Z}$, as the set of points whose indices belong to the set

$$\mathcal{N}^v_{(i,j)} \triangleq \left\{ (k,l) \;\middle|\; |k - i| + |l - j| \leq 1,\ k, l \in \mathbb{Z} \right\}. \tag{5.18c}$$

The gray points in Figure 5.1 are $\mathcal{N}^v_{(1,2)}$. For simplicity, we round in GenOpt the user-specified number of particles $n'_P \in \mathbb{N}$ to the next biggest integer n_P

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

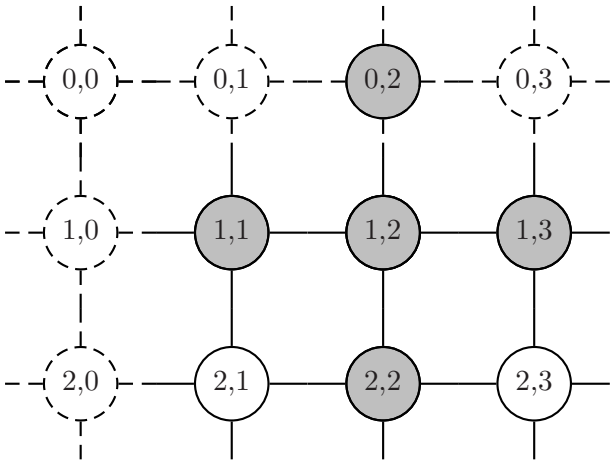


Figure 5.1: Section of a 2-dimensional lattice of particles with $\sqrt{n}_P \geq 3$. The particles belonging to the von Neumann neighborhood $\mathcal{N}^v_{(1,2)}$ with range 1, defined in (5.18c)*, are colored gray. Indicated by dashes are the particles that are generated by wrapping the indices.*

such that $\sqrt{n}_P \in \mathbb{N}$ and $n_P \geq n'_P$.[2] Then, we can wrap the indices by replacing, for $k \in \mathbb{Z}$, $(0, k)$ by $(\sqrt{n}_P, k)$, $(\sqrt{n}_P + 1, k)$ by $(1, k)$, and similarly by replacing $(k, 0)$ by $(k, \sqrt{n}_P)$ and $(k, \sqrt{n}_P + 1)$ by $(k, 1)$. Then, a particle with indices (k, l), with $1 \leq k \leq \sqrt{n}_P$ and $1 \leq l \leq \sqrt{n}_P$, has in the PSO algorithm the index $i = (k-1)\sqrt{n}_P + l$, and hence $i \in \{1, \ldots, n_P\}$.

Kennedy and Mendes [KM02] show that greater connectivity of the particles speeds up convergence, but it does not tend to improve the population's ability to discover the global optimum. Best performance has been achieved with the von Neumann topology, whereas neither the *gbest* nor the *lbest* topology seemed especially good in comparison with other topologies.

Carlisle and Dozier [CD01] achieve on unimodal and multi-modal functions for the *gbest* topology better results than for the *lbest* topology.

[2] In principle, the lattice need not be a square, but we do not see any computational disadvantage of selecting a square lattice.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

b) Model PSO Algorithm

We will now present the Model PSO Algorithm that is implemented in GenOpt.

Algorithm 5.4.1 (Model PSO Algorithm for Continuous Variables)

Data:	Constraint set $\mathbf{X}$, as defined in (4.1), but with finite lower and upper bound for each independent variable. Initial iterate $x_0 \in \mathbf{X}$. Number of particles $n_P \in \mathbb{N}$ and number of generations $n_G \in \mathbb{N}$.
Step 0:	Initialize $k = 0$, $x_0(0) = x_0$ and the neighborhoods $\{\mathcal{N}_i\}_{i=1}^{n_P}$.
Step 1:	Initialize $\{x_i(0)\}_{i=2}^{n_P} \subset \mathbf{X}$ randomly distributed.
Step 2:	For $i \in \{1, \ldots, n_P\}$, determine the local best particles $$p_{l,i}(k) \triangleq \underset{x \in \{x_i(m)\}_{m=0}^{k}}{\arg\min} \quad f(x) \tag{5.19a}$$ and the global best particle $$p_{g,i}(k) \triangleq \underset{x \in \{x_j(m) \mid j \in \mathcal{N}_i\}_{m=0}^{k}}{\arg\min} \quad f(x). \tag{5.19b}$$
Step 3:	Update the particle location $\{x_i(k+1)\}_{i=1}^{n_P} \subset \mathbf{X}$.
Step 4:	If $k = n_G$, stop. Else, go to Step 2.
Step 5:	Replace k by $k+1$, and go to Step 1.

We will now discuss the different implementations of the Model PSO Algorithm 5.4.1 in GenOpt.

c) Particle Update Equation

(i) Version with Inertia Weight Eberhart and Shi [SE98, SE99] introduced an *inertia weight* $w(k)$ which improves the performance of the original PSO algorithm. In the version with inertia weight, the particle update equation is, for all $i \in \{1, \ldots, n_P\}$, for $k \in \mathbb{N}$ and $x_i(k) \in \mathbb{R}^{n_c}$, with $v_i(0) = 0$,

$$\begin{aligned} \widehat{v}_i(k+1) &= w(k)\, v_i(k) + c_1\, \rho_1(k) \big(p_{l,i}(k) - x_i(k)\big) \\ &\quad + c_2\, \rho_2(k) \big(p_{g,i}(k) - x_i(k)\big), && (5.20a) \\ v_i^j(k+1) &= \operatorname{sign}(\widehat{v}_i^j(k+1)) \min\{|\widehat{v}_i^j(k+1)|, v_{max}^j\}, \\ &\qquad j \in \{1, \ldots, n_c\}, && (5.20b) \\ x_i(k+1) &= x_i(k) + v_i(k+1), && (5.20c) \end{aligned}$$

where

$$v_{max}^j \triangleq \lambda\, (u^j - l^j), \tag{5.20d}$$

with $\lambda \in \mathbb{R}_+$, for all $j \in \{1, \ldots, n_c\}$, and $l, u \in \mathbb{R}^{n_c}$ are the lower and upper bound of the independent variable. A common value is $\lambda = 1/2$. In GenOpt, if $\lambda \leq 0$, then no velocity clamping is used, and hence, $v_i^j(k+1) = \widehat{v}_i^j(k+1)$, for all $k \in \mathbb{N}$, all $i \in \{1, \ldots, n_P\}$ and all $j \in \{1, \ldots, n_c\}$.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

We compute the inertia weight as

$$w(k) = w_0 - \frac{k}{K}\,(w_0 - w_1), \tag{5.20e}$$

where $w_0 \in \mathbb{R}$ is the initial inertia weight, $w_1 \in \mathbb{R}$ is the inertia weight for the last generation, with $0 \le w_1 \le w_0$, and $K \in \mathbb{N}$ is the maximum number of generations. $w_0 = 1.2$ and $w_1 = 0$ can be considered as good choices [PV02b].

(ii) Version with Constriction Coefficient Clerc and Kennedy [CK02] introduced a version with a constriction coefficient that reduces the velocity. In their Type 1" implementation, the particle update equation is, for all $i \in \{1, \ldots, n_P\}$, for $k \in \mathbb{N}$ and $x_i(k) \in \mathbb{R}^{n_c}$, with $v_i(0) = 0$,

$$\begin{aligned}
\widehat{v}_i(k+1) &= \chi(\kappa,\varphi)\,\big(v_i(k) + c_1\,\rho_1(k)\,\big(p_{l,i}(k) - x_i(k)\big) \\
&\quad + c_2\,\rho_2(k)\,\big(p_{g,i}(k) - x_i(k)\big)\big), && (5.21a)\\
v_i^j(k+1) &= \operatorname{sign}(\widehat{v}_i^j(k+1))\,\min\{|\widehat{v}_i^j(k+1)|, v_{max}^j\}, \\
&\qquad j \in \{1, \ldots, n_c\}, && (5.21b)\\
x_i(k+1) &= x_i(k) + v_i(k+1), && (5.21c)
\end{aligned}$$

where

$$v_{max}^j \triangleq \lambda\,(u^j - l^j), \tag{5.21d}$$

is as in (5.20d).

In (5.21a), $\chi(\kappa,\varphi)$ is called *constriction coefficient*, defined as

$$\chi(\kappa,\varphi) \triangleq \begin{cases} \frac{2\,\kappa}{|2-\varphi-\sqrt{\varphi^2-4\,\varphi}|}, & \text{if } \varphi > 4, \\ \kappa, & \text{otherwise,} \end{cases} \tag{5.21e}$$

where $\varphi \triangleq c_1 + c_2$ and $\kappa \in (0, 1]$ control how fast the population collapses into a point. If $\kappa = 1$, the space is thoroughly searched, which yields slower convergence.

Equation (5.21) can be used with or without velocity clamping (5.21b). If velocity clamping (5.21b) is used, Clerc and Kennedy use $\varphi = 4.1$, otherwise they use $\varphi = 4$. In either case, they set $c_1 = c_2 = \varphi/2$ and a population size of $n_P = 20$.

Carlisle and Dozier [CD01] recommend the settings $n_P = 30$, no velocity clamping, $\kappa = 1$, $c_1 = 2.8$ and $c_2 = 1.3$.

Kennedy and Eberhart [KES01] report that using velocity clamping (5.21b) and a constriction coefficient shows faster convergence for some test problems compared to using an inertia weight, but the algorithm tends to get stuck in local minima.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

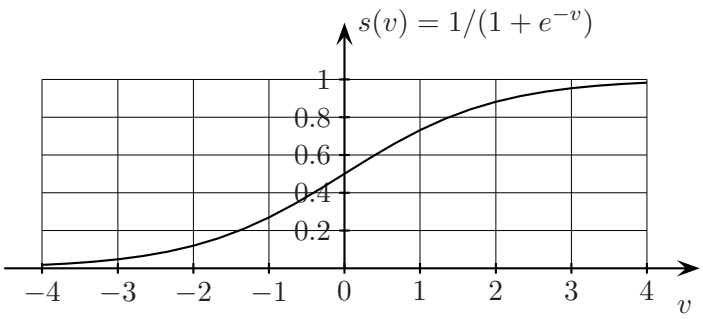


Figure 5.2: Sigmoid function.

5.4.2 PSO for Discrete Variables

Kennedy and Eberhart [KE97] introduced a binary version of the PSO algorithm to solve problem $\mathbf{P}_d$ defined in (4.4).

The binary PSO algorithm encodes the discrete independent variables in a string of binary numbers and then operates with this binary string. For some $i \in \{1, \ldots, n_d\}$, let $x_i \in \mathbb{N}$ be the component of a discrete independent variable, and let $\psi_i \in \{0,1\}^{m_i}$ be its binary representation (with $m_i \in \mathbb{N}_+$ bits), obtained using Gray encoding [PFTV93], and let $\pi_{l,i}(k)$ and $\pi_{g,i}(k)$ be the binary representation of $p_{l,i}(k)$ and $p_{g,i}(k)$, respectively, where $p_{l,i}(k)$ and $p_{g,i}(k)$ are defined in (5.19).

Then, for $i \in \{1, \ldots, n_d\}$ and $j \in \{1, \ldots, m_i\}$ we initialize randomly $\psi_i^j(0) \in \{0,1\}$, and compute, for $k \in \mathbb{N}$,

$$\begin{aligned} \widehat{v}_i^j(k+1) &= v_i^j(k) + c_1\,\rho_1(k)\left(\pi_{l,i}^j(k) - \psi_i^j(k)\right) \\ &\quad + c_2\,\rho_2(k)\left(\pi_{g,i}^j(k) - \psi_i^j(k)\right)\Big), \qquad (5.22a) \\ v_i^j(k+1) &= \operatorname{sign}(\widehat{v}_i^j(k+1))\,\min\{|\widehat{v}_i^j(k+1)|, v_{max}\}, \qquad (5.22b) \\ \psi_i^j(k+1) &= \begin{cases} 0, & \text{if } \rho_{i,j}(k) \geq s\big(v_i^j(k+1)\big), \\ 1, & \text{otherwise,} \end{cases} \qquad (5.22c) \end{aligned}$$

where

$$s(v) \triangleq \frac{1}{1+e^{-v}} \qquad (5.22d)$$

is the sigmoid function shown in Fig. 5.2 and $\rho_{i,j}(k) \sim U(0,1)$, for all $i \in \{1, \ldots, n_d\}$ and for all $j \in \{1, \ldots, m_i\}$.

In (5.22b), $v_{max} \in \mathbb{R}_+$ is often set to 4 to prevent a saturation of the sigmoid function, and $c_1, c_2 \in \mathbb{R}_+$ are often such that $c_1 + c_2 = 4$ (see [KES01]).

Notice that $s(v) \to 0.5$, as $v \to 0$, and consequently the probability of flipping a bit goes to 0.5. Thus, in the binary PSO, a small v_{max} causes a large exploration, whereas in the continuous PSO, a small v_{max} causes a small exploration of the search space.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Any of the above neighborhood topologies can be used, and Model Algorithm 5.4.1 applies if we replace the constraint set $\mathbf{X}$ by the user-specified set $\mathbf{X}_d \subset \mathbb{Z}^{n_d}$.

5.4.3 PSO for Continuous and Discrete Variables

For problem $\mathbf{P}_{cd}$ defined in (4.6), we treat the continuous independent variables as in (5.20) or (5.21), and the discrete independent variables as in (5.22). Any of the above neighborhood topologies can be used, and Model Algorithm 5.4.1 applies if we define the constraint set $\mathbf{X}$ as in (4.5).

5.4.4 PSO on a Mesh

We now present a modification to the previously discussed PSO algorithms. For evaluating the cost function, we will modify the continuous independent variables such that they belong to a fixed mesh in $\mathbb{R}^{n_c}$. Since the iterates of PSO algorithms typically cluster during the last iterations, this reduces in many cases the number of simulation calls during the optimization. The modification is done by replacing the cost function $f\colon \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} \to \mathbb{R}$ in Model Algorithm 5.4.1 as follows: Let $x_0 \triangleq (x_{c,0}, x_{d,0}) \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_c}$ denote the initial iterate, let $\mathbf{X}_c$ be the feasible set for the continuous independent variables defined in (4.5b), let $r, s \in \mathbb{N}$, with $r > 1$, be user-specified parameters, let

$$\Delta \triangleq \frac{1}{r^s} \tag{5.23}$$

and let the mesh be defined as

$$\mathbb{M}(x_{c,0}, \Delta, s) \triangleq \left\{ x_{c,0} + \Delta \sum_{i=1}^{n} m^i \, s^i \, e_i \mid m \in \mathbb{Z}^{n_c} \right\}, \tag{5.24}$$

where $s \in \mathbb{R}^{n_c}$ is equal to the value defined by the variable `Step` in GenOpt's command file (see page 83). Then, we replace $f(\cdot,\cdot)$ by $\widehat{f}\colon \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} \times \mathbb{R}^{n_c} \times \mathbb{R} \times \mathbb{R}^{n_c} \to \mathbb{R}$, defined by

$$\widehat{f}(x_c, x_d; x_{c,0}, \Delta, s) \triangleq f(\gamma(x_c), x_d), \tag{5.25}$$

where $\gamma\colon \mathbb{R}^{n_c} \to \mathbb{R}^{n_c}$ is the projection of the continuous independent variable to the closest feasible mesh point, i.e., $\gamma(x_c) \in \mathbb{M}(x_{c,0}, \Delta, s) \cap \mathbf{X}_c$. Thus, for evaluating the cost function, the continuous independent variables are replaced by the closest feasible mesh point, and the discrete independent variables remain unchanged.

Good numerical results have been obtained by selecting $s \in \mathbb{R}^{n_c}$ and $r, s \in \mathbb{N}$ such that about 50 to 100 mesh points are located along each coordinate direction.

5.4.5 Population Size and Number of Generations

Parsopoulos and Vrahatis [PV02b] use for $x \in \mathbb{R}^{n_c}$ a population size of about $5\,n$ up to $n = 15$. For $n \approx 10 \ldots 20$, they use $n_P \approx 10\,n$. They set

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

the number of generations to $n_G = 1000$ up to $n = 20$ and to $n_G = 2000$ for $n = 30$.

Van den Bergh and Engelbrecht [vdBE01] recommend using more than 20 particles and 2000 to 5000 generations.

Kennedy and Eberhart [KES01] use, for test cases with the *lbest* neighborhood topology of size $l = 2$ and $n = 2$ and $n = 30$, a population size of $n_P = 20 \ldots 30$. They report that $10 \ldots 50$ particles usually work well. As a rule of thumb, they recommend for the *lbest* neighborhood to select the neighborhood size such that each neighborhood consists of $10 \ldots 20\%$ of the population.

5.4.6 Keywords

For the Particle Swarm algorithm, the command file (see page 83) can contain continuous and discrete independent variables.

The different specifications for the `Algorithm` section of the GenOpt command file are as follows:

PSO algorithm with inertia weight:

```
Algorithm{
  Main                      = PSOIW;
  NeighborhoodTopology      = gbest | lbest | vonNeumann;
  NeighborhoodSize          = Integer;  // 0 < NeighborhoodSize
  NumberOfParticle          = Integer;
  NumberOfGeneration        = Integer;
  Seed                      = Integer;
  CognitiveAcceleration     = Double;   // 0 < CognitiveAcceleration
  SocialAcceleration        = Double;   // 0 < SocialAcceleration
  MaxVelocityGainContinuous = Double;
  MaxVelocityDiscrete       = Double;   // 0 < MaxVelocityDiscrete
  InitialInertiaWeight      = Double;   // 0 < InitialInertiaWeight
  FinalInertiaWeight        = Double;   // 0 < FinalInertiaWeight
}
```

PSO algorithm with constriction coefficient:

```
Algorithm{
  Main                      = PSOCC;
  NeighborhoodTopology      = gbest | lbest | vonNeumann;
  NeighborhoodSize          = Integer;  // 0 < NeighborhoodSize
  NumberOfParticle          = Integer;
  NumberOfGeneration        = Integer;
  Seed                      = Integer;
  CognitiveAcceleration     = Double;   // 0 < CognitiveAcceleration
  SocialAcceleration        = Double;   // 0 < SocialAcceleration
  MaxVelocityGainContinuous = Double;
  MaxVelocityDiscrete       = Double;   // 0 < MaxVelocityDiscrete
```

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

```
  ConstrictionGain          = Double;   // 0 < ConstrictionGain <= 1
}
```

PSO algorithm with constriction coefficient and continuous independent variables restricted to a mesh:

```
Algorithm{
  Main                      = PSOCCMesh;
  NeighborhoodTopology      = gbest | lbest | vonNeumann;
  NeighborhoodSize          = Integer;  // 0 < NeighborhoodSize
  NumberOfParticle          = Integer;
  NumberOfGeneration        = Integer;
  Seed                      = Integer;
  CognitiveAcceleration     = Double;   // 0 < CognitiveAcceleration
  SocialAcceleration        = Double;   // 0 < SocialAcceleration
  MaxVelocityGainContinuous = Double;
  MaxVelocityDiscrete       = Double;   // 0 < MaxVelocityDiscrete
  ConstrictionGain          = Double;   // 0 < ConstrictionGain <= 1
  MeshSizeDivider           = Integer;  // 1 < MeshSizeDivider
  InitialMeshSizeExponent   = Integer;  // 0 <= InitialMeshSizeExponent
}
```

The entries that are common to all implementations are defined as follows:

`Main` The name of the main algorithm. The implementation `PSOIW` uses the location update equation (5.20) for the continuous independent variables, and the implementation `PSOCC` uses (5.21) for the continuous independent variables. All implementations use (5.22) for the discrete independent variables.

`NeighborhoodTopology` This entry defines what neighborhood topology is being used.

`NeighborhoodSize` For the *lbest* neighborhood topology, this entry is equal to l in (5.18a). For the *gbest* and the *von Neumann* neighborhood topology, the value of `NeighborhoodSize` is ignored.

`NumberOfParticle` This is equal to the variable $n_P \in \mathbb{N}$.

`NumberOfGeneration` This is equal to the variable $n_G \in \mathbb{N}$ in Algorithm 5.4.1.

`Seed` This value is used to initialize the random number generator.

`CognitiveAcceleration` This is equal to the variable $c_1 \in \mathbb{R}_+$.

`SocialAcceleration` This is equal to the variable $c_2 \in \mathbb{R}_+$.

`MaxVelocityGainContinuous` This is equal to the variable $\lambda \in \mathbb{R}_+$ in (5.20d) and in (5.21d). If `MaxVelocityGainContinuous` is set to zero or to a negative value, then no velocity clamping is used, and hence, $v_i^j(k+1) = \widehat{v}_i^j(k+1)$, for all $k \in \mathbb{N}$, all $i \in \{1, \ldots, n_P\}$ and all $j \in \{1, \ldots, n_c\}$.

`MaxVelocityDiscrete` This is equal to the variable $v_{max} \in \mathbb{R}_+$ in (5.22b).

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

For the `PSOIW` implementation, following additional entries must be specified:

`InitialInertiaWeight` This is equal to $w_0 \in \mathbb{R}_+$ in (5.20e).

`FinalInertiaWeight` This is equal to $w_1 \in \mathbb{R}_+$ in (5.20e).

For the `PSOCC` implementation, following additional entries must be specified:

`ConstrictionGain` This is equal to $\kappa \in (0, 1]$ in (5.21e).

Notice that for discrete independent variables, the entries of `InitialInertiaWeight`, `FinalInertiaWeight`, and `ConstrictionGain` are ignored.

For the `PSOCCMesh` implementation, following additional entries must be specified:

`MeshSizeDivider` This is equal to $r \in \mathbb{N}$, with $r > 1$, used in (5.23).

`InitialMeshSizeExponent` This is equal to $s \in \mathbb{N}$ used in (5.23).

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

5.5 Hybrid Generalized Pattern Search Algorithm with Particle Swarm Optimization Algorithm

This hybrid global optimization algorithm can be used to solve problem $\mathbf{P}_c$ defined in (4.2) and problem $\mathbf{P}_{cd}$ defined in (4.6). Problem $\mathbf{P}_{cg}$ defined in (4.3) and problem $\mathbf{P}_{cdg}$ defined in (4.7) can be solved if the constraint functions $g(\cdot)$ are implemented as described in Section 8.2.

This hybrid global optimization algorithm starts by doing a Particle Swarm Optimization (PSO) on a mesh, as described in Section 5.4.4, for a user-specified number of generations $n_G \in \mathbb{N}$. Afterwards, it initializes the Hooke-Jeeves Generalized Pattern Search (GPS) algorithm, described in Section 5.2.2, using the continuous independent variables of the particle with the lowest cost function value. If the optimization problem has continuous and discrete independent variables, then the discrete independent variables will for the GPS algorithm be fixed at the value of the particle with the lowest cost function value.

We will now explain the hybrid algorithm for the case where all independent variables are continuous, and then for the case with mixed continuous and discrete independent variables. Throughout this section, we will denote the dimension of the continuous independent variables by $n_c \in \mathbb{N}$ and the dimension of the discrete independent variables by $n_d \in \mathbb{N}$.

5.5.1 Hybrid Algorithm for Continuous Variables

We will now discuss the hybrid algorithm to solve problem $\mathbf{P}_c$ defined in (4.2). However, we require the constraint set $\mathbf{X} \subset \mathbb{R}^{n_c}$ defined in (4.1) to have finite lower and upper bounds $l^i, u^i \in \mathbb{R}$, for all $i \in \{1, \ldots, n_c\}$.

First, we run the PSO algorithm 5.4.1, with user-specified initial iterate $x_0 \in \mathbf{X}$ for a user-specified number of generations $n_G \in \mathbb{N}$ on the mesh defined in (5.24). Afterwards, we run the GPS algorithm 5.1.8 where the initial iterate x_0 is equal to the location of the particle with the lowest cost function value, i.e.,

$$x_0 \triangleq p \triangleq \underset{x \in \{x_j(k) \ \mid \ j \in \{1,\ldots,n_P\},\ k \in \{1,\ldots,n_G\}\}}{\arg\min} f(x), \tag{5.26}$$

where $n_P \in \mathbb{N}$ denotes the number of particles and $x_j(k)$, $j \in \{1, \ldots, n_P\}$, $k \in \{1, \ldots, n_G\}$ are as in Algorithm 5.4.1.

Since the PSO algorithm terminates after a finite number of iterations, all convergence results of the GPS algorithm hold. In particular, if the cost function is once continuously differentiable, then the hybrid algorithm constructs accumulation points that are feasible stationary points of problem (4.2) (see Theorem 5.1.13).

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Since the PSO algorithm is a global optimization algorithm, the hybrid algorithm is, compared to the Hooke-Jeeves algorithm, less likely to be attracted by a local minimum that is not global. Thus, the hybrid algorithm combines the global features of the PSO algorithm with the provable convergence properties of the GPS algorithm.

If the cost function is discontinuous, then the hybrid algorithm is, compared to the Hooke-Jeeves algorithm, less likely to jam at a discontinuity far from a solution.

5.5.2 Hybrid Algorithm for Continuous and Discrete Variables

For problem $\mathbf{P}_{cd}$ defined in (4.6) with continuous and discrete independent variables, we run the PSO algorithm 5.4.1, with user-specified initial iterate $x_0 \in \mathbf{X} \triangleq \mathbf{X}_c \times \mathbf{X}_d \subset \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}$ for a user-specified number of generations $n_G \in \mathbb{N}$, where the continuous independent variables are restricted to the mesh defined in (5.24). We require the constraint set $\mathbf{X}_c \subset \mathbb{R}^{n_c}$ defined in (4.5b) to have finite lower and upper bounds $l^i, u^i \in \mathbb{R}$, for all $i \in \{1, \ldots, n_c\}$.

Afterwards, we run the GPS algorithm 5.1.8, where the initial iterate $x_0 \in \mathbf{X}_c$ is equal to $p_c \in \mathbf{X}_c$, which we define as the continuous independent variables of the particle with the lowest cost function value, i.e., $p \triangleq (p_c, p_d) \in \mathbf{X}_c \times \mathbf{X}_d$, where p is defined in (5.26). In the GPS algorithm, we fix the discrete components at $p_d \in \mathbf{X}_d$ for all iterations. Thus, we use the GPS algorithm to refine the continuous components of the independent variables, and fix the discrete components of the independent variables.

5.5.3 Keywords

For this algorithm, the command file (see page 83) can contain continuous and discrete independent variables. It must contain at least one continuous parameter.

The specifications of the `Algorithm` section of the GenOpt command file is as follows:

Note that the first entries are as for the PSO algorithm on page 39 and the last entries are as for GPS implementation of the Hooke-Jeeves algorithm on page 24.

```
Algorithm{
  Main                      = GPSPSOCCHJ;
  NeighborhoodTopology      = gbest | lbest | vonNeumann;
  NeighborhoodSize          = Integer;  // 0 <  NeighborhoodSize
  NumberOfParticle          = Integer;
  NumberOfGeneration        = Integer;
  Seed                      = Integer;
```

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

```
  CognitiveAcceleration     = Double;   // 0 <  CognitiveAcceleration
  SocialAcceleration        = Double;   // 0 <  SocialAcceleration
  MaxVelocityGainContinuous = Double;
  MaxVelocityDiscrete       = Double;   // 0 <  MaxVelocityDiscrete
  ConstrictionGain          = Double;   // 0 <  ConstrictionGain <= 1
  MeshSizeDivider           = Integer;  // 1 <  MeshSizeDivider
  InitialMeshSizeExponent   = Integer;  // 0 <= InitialMeshSizeExponent
  MeshSizeExponentIncrement = Integer;  // 0 <  MeshSizeExponentIncrement
  NumberOfStepReduction     = Integer;  // 0 <  NumberOfStepReduction
}
```

The entries are defined as follows:

`Main` The name of the main algorithm.

`NeighborhoodTopology` This entry defines what neighborhood topology is being used.

`NeighborhoodSize` This entry is equal to l in (5.18). For the *gbest* neighborhood topology, the value of `NeighborhoodSize` will be ignored.

`NumberOfParticle` This is equal to the variable $n_P \in \mathbb{N}$.

`NumberOfGeneration` This is equal to the variable $n_G \in \mathbb{N}$ in Algorithm 5.4.1.

`Seed` This value is used to initialize the random number generator.

`CognitiveAcceleration` This is equal to the variable $c_1 \in \mathbb{R}_+$ used by the PSO algorithm.

`SocialAcceleration` This is equal to the variable $c_2 \in \mathbb{R}_+$ used by the PSO algorithm.

`MaxVelocityGainContinuous` This is equal to the variable $\lambda \in \mathbb{R}_+$ in (5.20d) and in (5.21d). If `MaxVelocityGainContinuous` is set to zero or to a negative value, then no velocity clamping is used, and hence, $v_i^j(k+1) = \widehat{v}_i^j(k+1)$, for all $k \in \mathbb{N}$, all $i \in \{1, \ldots, n_P\}$ and all $j \in \{1, \ldots, n_c\}$.

`MaxVelocityDiscrete` This is equal to the variable $v_{max} \in \mathbb{R}_+$ in (5.22b).

`ConstrictionGain` This is equal to $\kappa \in (0, 1]$ in (5.21e).

`MeshSizeDivider` This is equal to $r \in \mathbb{N}$, with $r > 1$, used by the PSO algorithm in (5.23) and used by the GPS algorithm to compute $\Delta_k \triangleq 1/r^{s_k}$ (see equation (5.6a)). A common value is $r = 2$.

`InitialMeshSizeExponent` This is equal to $s \in \mathbb{N}$ used by the PSO algorithm in (5.23) and used by the GPS algorithm in (5.6a). A common value is $s_0 = 0$.

`MeshSizeExponentIncrement` The value for $t_k \in \mathbb{N}$ (fixed for all $k \in \mathbb{N}$) used by the GPS algorithm in (5.6a). A common value is $t_k = 1$.

`NumberOfStepReduction` The maximum number of step reductions before the GPS algorithm stops. Thus, if we use the notation $m \triangleq$ `NumberOfStepReduction`, then we have for the last iterations $\Delta_k = 1/r^{s_0 + m\, t_k}$. A common value is $m = 4$.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

5.6 Simplex Algorithm of Nelder and Mead with the Extension of O'Neill

The Simplex algorithm of Nelder and Mead is a derivative free optimization algorithm. It can be used to seek a solution of problem $\mathbf{P}_c$ defined in (4.2) and problem $\mathbf{P}_{cg}$ defined in (4.3), with constraints on the dependent parameters implemented as described in Section 8. The number of independent parameters n must be larger than 1.

The Simplex algorithm constructs an n-dimensional simplex in the space that is spanned by the independent parameters. At each of the $(n+1)$ vertices of the simplex, the value of the cost function is evaluated. In each iteration step, the point with the highest value of the cost function is replaced by another point. The algorithm consists of three main operations: (a) *point reflection*, (b) *contraction of the simplex* and (c) *expansion of the simplex*.

Despite the well known fact that the Simplex algorithm can fail to converge to a stationary point [Kel99b, Tor89, Kel99a, Wri96, McK98, LRWW98], both in practice and theory, particularly if the dimension of independent variables is large, say bigger than 10 [Tor89], it is an often used algorithm. Several improvements to the Simplex algorithm or algorithms that were motivated by the Simplex algorithm exist, see for example [Kel99b, Tor89, Kel99a, Tse99]. However, in GenOpt, we use the original Nelder-Mead algorithm [NM65] with the extension of O'Neill [O'N71]. Optionally, the here implemented algorithm allows using a modified stopping criteria.

We will now explain the different steps of the Simplex algorithm.

5.6.1 Main Operations

The notation defined below is used in describing the main operations. The operations are illustrated in Fig. 5.3 where for simplicity a two-dimensional simplex is illustrated.

We now introduce some notation and definitions.

1. We will denote by $\mathbf{I} \triangleq \{1, \ldots, n+1\}$ the set of all vertex indices.
2. We will denote by $l \in \mathbf{I}$ the smallest index in $\mathbf{I}$ such that

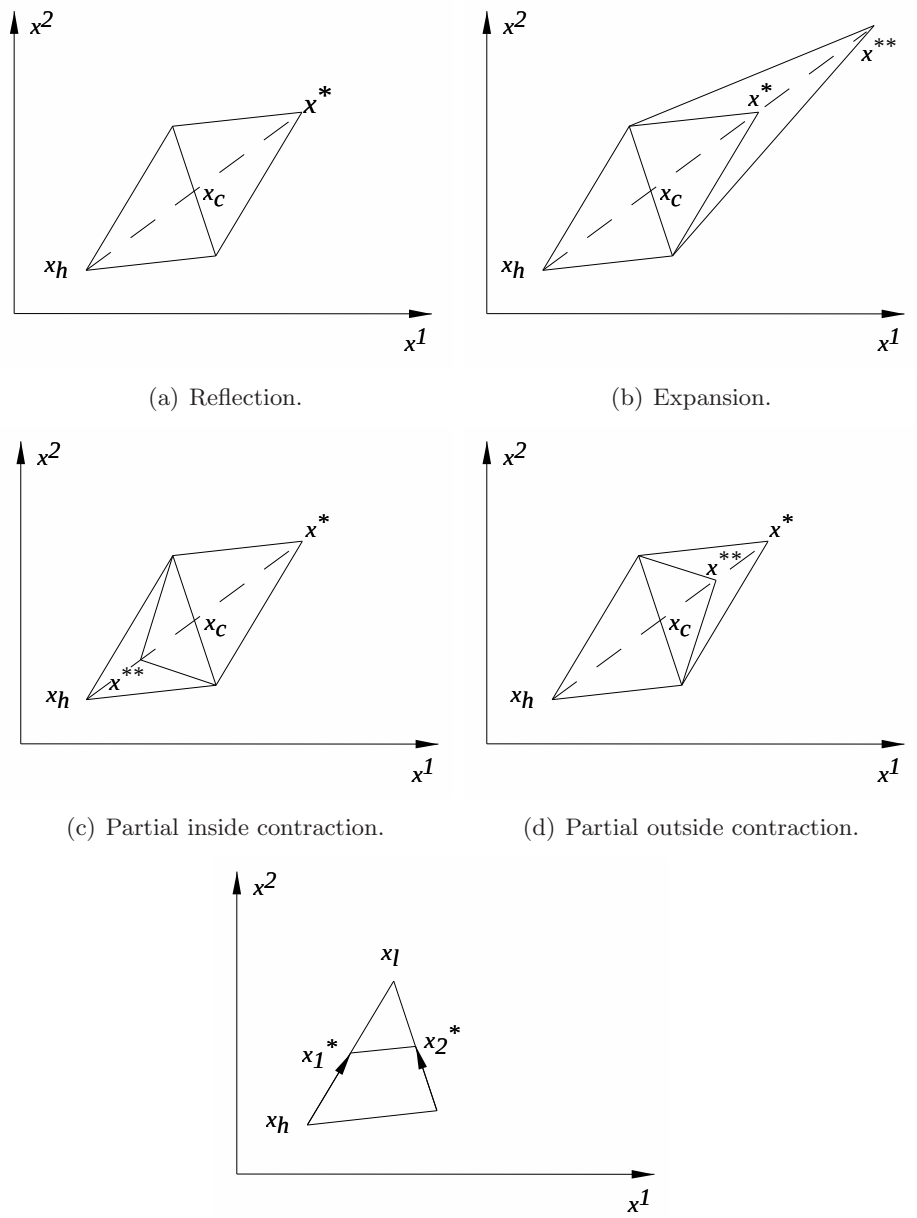
$$l = \arg\min_{i \in \mathbf{I}} f(x_i). \tag{5.27a}$$

   Hence, $f(x_l) \leq f(x_i)$, for all $i \in \mathbf{I}$.
3. We will denote by $h \in \mathbf{I}$ the smallest index in $\mathbf{I}$ such that

$$h = \arg\max_{i \in \mathbf{I}} f(x_i). \tag{5.27b}$$

   Hence, $f(x_h) \geq f(x_i)$, for all $i \in \mathbf{I}$.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.



(a) Reflection.

(b) Expansion.

(c) Partial inside contraction.

(d) Partial outside contraction.

(e) Total contraction.

Figure 5.3: Simplex operations.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

4. Let x_i, for $i \in \mathbf{I}$, denote the simplex vertices, and let h be as in (5.27b). We will denote by $x_c \in \mathbb{R}^n$ the *centroid* of the simplex, defined as

$$x_c \triangleq \frac{1}{n} \sum_{\substack{i=1 \\ i \neq h}}^{n+1} x_i \tag{5.27c}$$

Next, we introduce the three main operations.

Reflection Let $h \in \mathbf{I}$ be as in (5.27b) and let x_c be as in (5.27c). The reflection of $x_h \in \mathbb{R}^n$ to a point denoted as $x^* \in \mathbb{R}^n$ is defined as

$$x^* \triangleq (1+\alpha)\, x_c - \alpha\, x_h, \tag{5.28a}$$

where $\alpha \in \mathbb{R}$, with $\alpha > 0$, is called the *reflection coefficient*.

Expansion of the simplex Let $x^* \in \mathbb{R}^n$ be as in (5.28a) and x_c be as in (5.27c). The expansion of $x^* \in \mathbb{R}^n$ to a point denoted as $x^{**} \in \mathbb{R}^n$ is defined as

$$x^{**} \triangleq \gamma\, x^* + (1-\gamma)\, x_c, \tag{5.28b}$$

where $\gamma \in \mathbb{R}$, with $\gamma > 1$, is called the *expansion coefficient*.

Contraction of the simplex Let $h \in \mathbf{I}$ be as in (5.27b) and x_c be as in (5.27c). The contraction of $x_h \in \mathbb{R}^n$ to a point denoted as $x^{**} \in \mathbb{R}^n$ is defined as

$$x^{**} \triangleq \beta\, x_h + (1-\beta)\, x_c, \tag{5.28c}$$

where $\beta \in \mathbb{R}$, with $0 < \beta < 1$, is called the *contraction coefficient*.

5.6.2 Basic Algorithm

In this section, we describe the basic Nelder and Mead algorithm [NM65]. The extension of O'Neill and the modified restart criterion are discussed later. The algorithm is as follows:

1. Initialization: Given an initial iterate $x_1 \in \mathbb{R}^n$, a scalar c, with $c = 1$ in the initialization, a vector $s \in \mathbb{R}^n$ with user-specified step sizes for each independent parameter, and the set of unit coordinate vectors $\{e_i\}_{i=1}^n$, construct an initial simplex with vertices, for $i \in \{1, \ldots, n\}$,

$$x_{i+1} = x_1 + c\, s^i\, e_i. \tag{5.29}$$

   Compute $f(x_i)$, for $i \in \mathbf{I}$.

2. Reflection: Reflect the worst point, that is, compute x^* as in (5.28a).

3. Test whether we got the best point: If $f(x^*) < f(x_l)$, expand the simplex using (5.28b) since further improvement in this direction is likely. If $f(x^{**}) < f(x_l)$, then x_h is replaced by x^{**}, otherwise x_h is replaced by x^*, and the procedure is restarted from 2.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

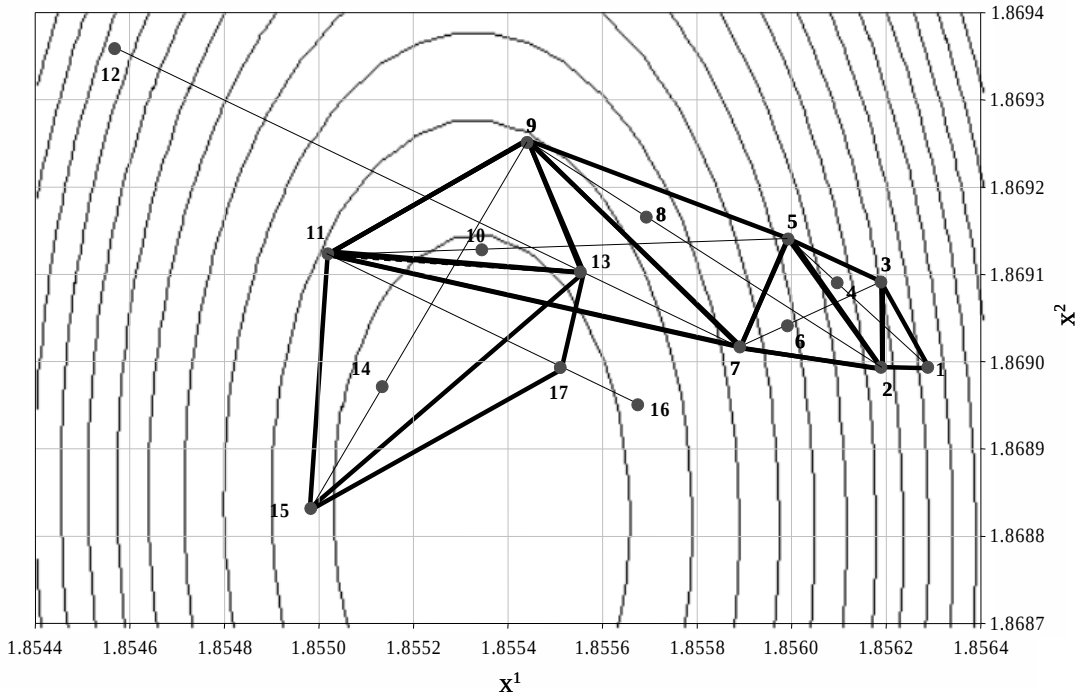


Figure 5.4: Sequence of iterates generated by the Simplex algorithm.

4. If it turned out under 3 that $f(x^*) \geq f(x_l)$, then we check if the new point x^* is the worst of all points: If $f(x^*) > f(x_i)$, for all $i \in \mathbf{I}$, with $i \neq h$, we contract the simplex (see 5); otherwise we replace x_h by x^* and go to 2.

5. For the contraction, we first check if we should try a partial outside contraction or a partial inside contraction: If $f(x^*) \geq f(x_h)$, then we try a partial inside contraction. To do so, we leave our indices as is and apply (5.28c). Otherwise, we try a partial outside contraction. This is done by replacing x_h by x^* and applying (5.28c). After the partial inside or the partial outside contraction, we continue at 6.

6. If $f(x^{**}) \geq f(x_h)$[3], we do a total contraction of the simplex by replacing $x_i \leftarrow (x_i + x_l)/2$, for all $i \in \mathbf{I}$. Otherwise, we replace x_h by x^{**}. In both cases, we continue from 2.

[3]Nelder and Mead [NM65] use the strict inequality $f(x^{**}) > f(x_h)$. However, if the user writes the cost function value with only a few representative digits to a text file, then the function looks like a step function if slow convergence is achieved. In such cases, $f(x^{**})$ might sometimes be equal to $f(x_h)$. Experimentally, it has been shown advantageous to perform a total contraction rather than continuing with a reflection. Therefore, the strict inequality has been changed to a weak inequality.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Fig. 5.4 shows a contour plot of a cost function $f\colon \mathbb{R}^n \to \mathbb{R}$ with a sequence of iterates generated by the Simplex algorithm. The sequence starts with constructing an initial simplex x_1, x_2, x_3. x_1 has the highest function value and is therefore reflected, which generates x_4. x_4 is the best point in the set $\{x_1, x_2, x_3, x_4\}$. Thus, it is further expanded, which generates x_5. x_2, x_3 and x_5 now span the new simplex. In this simplex, x_3 is the vertex with the highest function value and hence goes over to x_6 and further to x_7. The process of reflection and expansion is continued again two times, which leads to the simplex spanned by x_7, x_9 and x_{11}. x_7 goes over to x_{12} which turns out to be the worst point. Hence, we do a partial inside contraction, which generates x_{13}. x_{13} is better than x_7 so we use the simplex spanned by x_9, x_{11} and x_{13} for the next reflection. The last steps of the optimization are for clarity not shown.

5.6.3 Stopping Criteria

The first criterion is a test of the variance of the function values at the vertices of the simplex

$$\frac{1}{n}\left(\sum_{i=1}^{n+1}\big(f(x_i)\big)^2 - \frac{1}{n+1}\left(\sum_{i=1}^{n+1} f(x_i)\right)^2\right) < \epsilon^2, \tag{5.30}$$

then the original implementation of the algorithm stops. Nelder and Mead have chosen this stopping criterion based on the statistical problem of finding the minimum of a sum of squares surface. In this problem, the curvature near the minimum yields information about the unknown parameters. A slight curvature indicates a high sampling variance of the estimate. Nelder and Mead argue that in such cases, there is no reason for finding the minimum point with high accuracy. However, if the curvature is marked, then the sampling variance is low and a higher accuracy in determining the optimal parameter set is desirable.

Note that the stopping criterion (5.30) requires the variance of the function values at the simplex vertices to be smaller than a prescribed limit. However, if $f(\cdot)$ has large discontinuities, which has been observed in building energy optimization problems [WW03], then the test (5.30) may never be satisfied. For this reason, among others, we do not recommend using this algorithm if the cost function has large discontinuities.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

5.6.4 O'Neill's Modification

O'Neill modified the termination criterion by adding a further condition [O'N71]. He checks whether any orthogonal step, each starting from the best vertex of the current simplex, leads to a further improvement of the cost function. He therefore sets $c = 0.001$ and tests if

$$f(x_l) < f(x) \tag{5.31a}$$

for all x defined by

$$x \triangleq x_l + c\, s^i\, e_i, \qquad i \in \{1, \ldots, n\}, \tag{5.31b}$$

where x_l denotes the best known point, and s^i and e_i are as in (5.29).

5.6.5 Modification of Stopping Criteria

In GenOpt, (5.31) has been modified. It has been observed that users sometimes write the cost function value with only few representative digits to the output file. In such cases, (5.31a) is not satisfied if the write statement in the simulation program truncates digits so that the difference $f(x_l) - f(x)$, where $f(\cdot)$ denotes the value that is read from the simulation output file, is zero. To overcome this numerical problem, (5.31b) has been modified to

$$x = x_l + \exp(j)\, c\, s^i\, e_i, \qquad i \in \{1, \ldots, n\} \tag{5.31c}$$

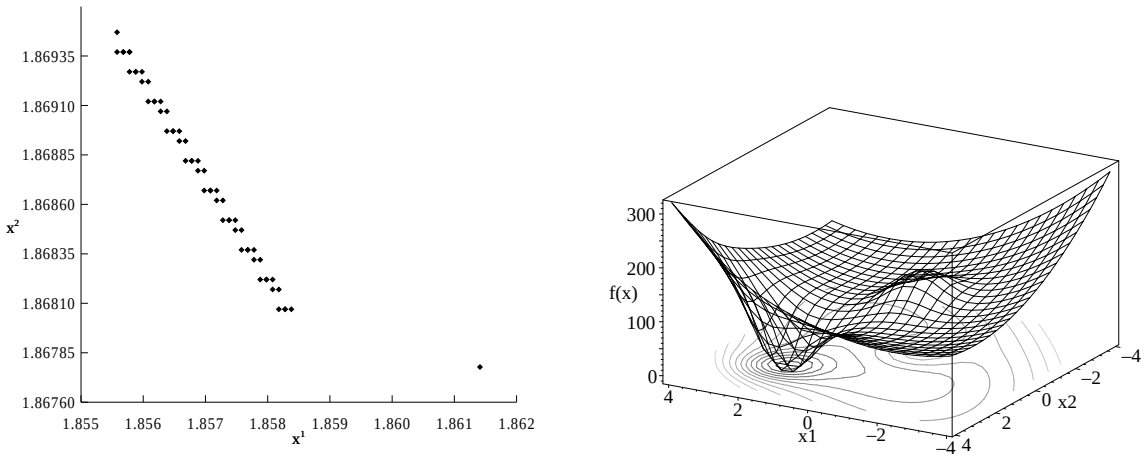
where for each direction $i \in \{1, \ldots, n\}$, the counter $j \in \mathbb{N}$ is set to zero for the first trial and increased by one as long as $f(x_l) = f(x)$.

If (5.31a) fails for any direction, then x computed by (5.31c) is the new starting point and a new simplex with side lengths $(c\, s^i)$, $i \in \{1, \ldots, n\}$, is constructed. The point x that failed (5.31a) is then used as the initial point x_l in (5.29).

Numerical experiments showed that during slow convergence the algorithm was restarted too frequently.

Fig. 5.5(a) shows a sequence of iterates where the algorithm was restarted too frequently. The iterates in the figure are part of the iteration sequence near the minimum of the test function shown in Fig. 5.5(b). The algorithm gets close to the minimum with appropriately large steps. The last of these steps can be seen at the right of the figure. After this step, the stopping criterion (5.30) was satisfied which led to a restart check, followed by a new construction of the simplex. From there on, the convergence was very slow due to the small step size. After each step, the stopping criterion was satisfied again which led to a new test of the optimality condition (5.31a), followed by a reconstruction of the simplex. This check is very costly in terms of function evaluations and, furthermore, the restart with a new simplex does not allow increasing the step size, though we are heading locally in the right direction.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.



(a) Sequence of iterates in the neighborhood of the minimum.

(b) 2-dimensional test function "2D1".

Figure 5.5: Nelder Mead trajectory.

O'Neill's modification prevents both excessive checking of the optimality condition as well as excessive reconstruction of the initial simplex. This is done by checking for convergence only after a predetermined number of steps (e.g., after five iterations). However, the performance of the algorithm depends strongly on this number. As an extreme case, a few test runs were done where convergence was checked after each step as in Fig. 5.5(a). It turned out that in some cases no convergence was reached within a moderate number of function evaluations if ϵ in (5.30) is chosen too large, e.g., $\epsilon = 10^{-3}$ (see Tab. 5.1).

To make the algorithm more robust, it is modified based on the following arguments:

1. If the simplex is moving in the same direction in the last two steps, then the search is not interrupted by checking for optimality since we are making steady progress in the moving direction.
2. If we do *not* have a partial inside or total contraction immediately beyond us, then it is likely that the minimum lies in the direction currently being explored. Hence, we do not interrupt the search with a restart.

These considerations have led to two criteria that both have to be satisfied to permit the convergence check according to (5.30), which might be followed by a check for optimality.

First, it is checked if we have done a partial inside contraction or a total contraction. If so, we check if the direction of the latest two steps in which the simplex is moving has changed by an angle of at least $(\pi/2)$. To do so, we

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

introduce the center of the simplex, defined by

$$x_m \triangleq \frac{1}{n+1} \sum_{i=1}^{n+1} x_i, \tag{5.32}$$

where x_i, $i \in \{1, \ldots, n\}$, are the simplex vertices. We also introduce the normalized direction of the simplex between two steps,

$$d_k \triangleq \frac{x_{m,k} - x_{m,k-1}}{\|x_{m,k} - x_{m,k-1}\|}, \tag{5.33}$$

where $k \in \mathbb{N}$ is the current iteration number.

We determine how much the simplex has changed its direction d_k between two steps by computing the inner product $\langle d_{k-1}, d_k \rangle$. The inner product is equal to the cosine of the angle d_{k-1} and d_k. If

$$\cos \phi_k = \langle d_{k-1},\, d_k \rangle \leq 0, \tag{5.34}$$

then the moving direction of the simplex has changed by at least $\pi/2$. Hence, the simplex has changed the exploration direction. Therefore, a minimum might be achieved and we need to test the variance of the vertices (5.30), possibly followed by a test of (5.31a).

Besides the above modification, a further modification was tested: In some cases, a reconstruction of the simplex after a failed check (5.31a) yields to slow convergence. Therefore, the algorithm was modified so that it continues at point 2 on page 46 without reconstructing the simplex after failing the test (5.31a). However, reconstructing the simplex led in most of the benchmark tests to faster convergence. Therefore, this modification is no longer used in the algorithm.

5.6.6 Benchmark Tests

Tab. 5.1 shows the number of function evaluations and Fig. 5.6 shows the relative number of function evaluations compared to the original implementation for several test cases. The different functions and the parameter settings are given in the Appendix. The only numerical parameter that was changed for the different optimizations is the accuracy, ϵ.

It turned out that modifying the stopping criterion is effective in most cases, particularly if a new simplex is constructed after the check (5.31a) failed. Therefore, the following two versions of the simplex algorithm are implemented in GenOpt:

1. The base algorithm of Nelder and Mead, including the extension of O'Neill. After failing (5.31a), the simplex is *always* reconstructed with the new step size.
2. The base algorithm of Nelder and Mead, including the extension of O'Neill, but with the modified stopping criterion as explained above. That is, the simplex is only reconstructed if its moving direction changed, and if we have an inside or total construction beyond us.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

	Accuracy							
	$\epsilon = 10^{-3}$				$\epsilon = 10^{-5}$			
Test function	Rosen-brock	2D1	Quad with I ma-trix	Quad with Q ma-trix	Rosen-brock	2D1	Quad with I ma-trix	Quad with Q ma-trix
Original, with recon-struction	137	120	3061	1075	139	109	1066	1165
Original, no recon-struction	136	110	1436	1356	139	109	1433	1253
Modified, with recon-struction	145	112	1296	1015	152	111	1060	1185
Modified, no recon-struction	155	120	1371	1347	152	109	1359	1312

Table 5.1: Comparison of the number of function evaluations for different implementations of the simplex algorithm. See Appendix for the definition of the function.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

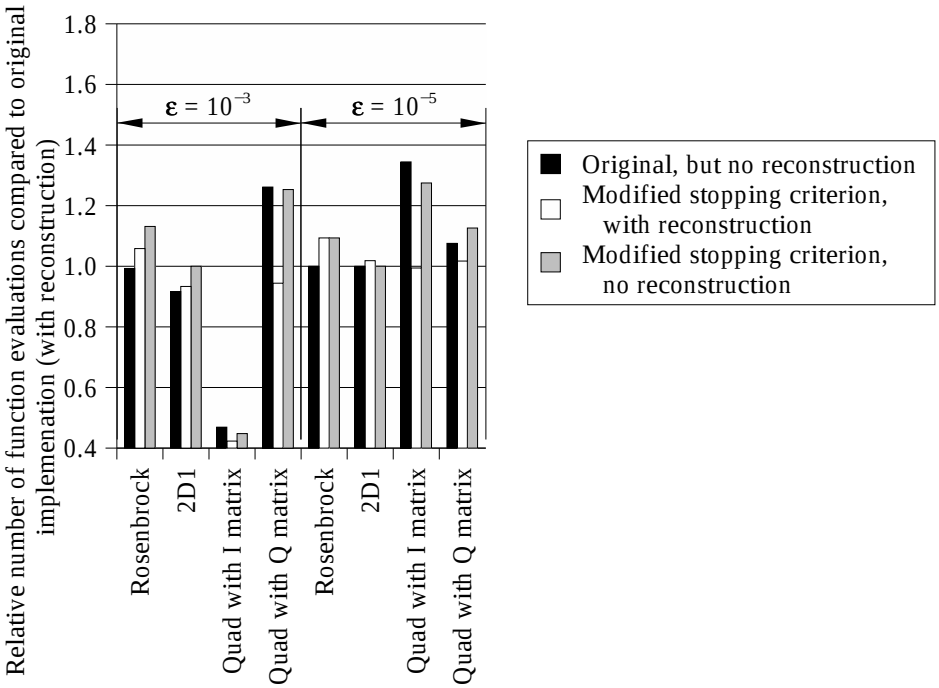


Figure 5.6: *Comparison of the benchmark tests.*

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

5.6.7 Keywords

For the Simplex algorithm, the command file (see page 83) must only contain continuous parameters.

To invoke the Simplex algorithm, the `Algorithm` section of the GenOpt command file must have following form:

```
Algorithm{
   Main                    = NelderMeadONeill;
   Accuracy                = Double;   // 0 <  Accuracy
   StepSizeFactor          = Double;   // 0 <  StepSizeFactor
   BlockRestartCheck       = Integer;  // 0 <= BlockRestartCheck
   ModifyStoppingCriterion = Boolean;
}
```

The key words have following meaning:

`Main` The name of the main algorithm.

`Accuracy` The accuracy that has to be reached before the optimality condition is checked. `Accuracy` is defined as equal to ϵ of (5.30), page 48.

`StepSizeFactor` A factor that multiplies the step size of each parameter for (a) testing the optimality condition and (b) reconstructing the simplex. `StepSizeFactor` is equal to c in (5.29) and (5.31c).

`BlockRestartCheck` Number that indicates for how many main iterations the restart criterion is not checked. If zero, restart might be checked after each main iteration.

`ModifyStoppingCriterion` Flag indicating whether the stopping criterion should be modified. If `true`, then the optimality check (5.30) is done only if both of the following conditions are satisfied: (a) in the last step, either a partial inside contraction or total contraction was done, and (b) the moving direction of the simplex has changed by an angle ϕ_k of at least $(\pi/2)$, where ϕ_k is computed using (5.34).

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

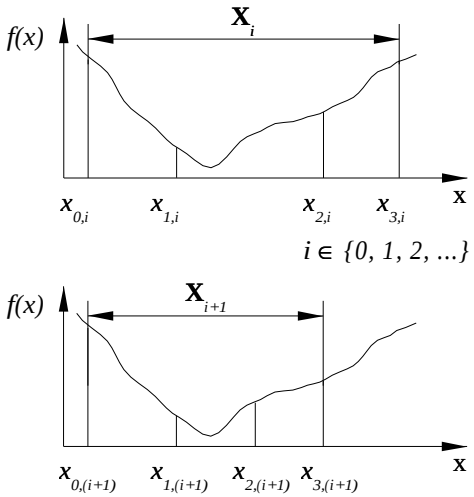


Figure 6.1: Interval division.

6 Algorithms for One-Dimensional Optimization

6.1 Interval Division Algorithms

Interval division algorithm can be used to minimize a function $f\colon \mathbb{R} \to \mathbb{R}$, (i.e., the function depends on one independent parameter only,) over a user-specified interval. The algorithms do not require derivatives and they require only one function evaluation per interval division, except for the initialization.

First, we explain a master algorithm for the interval division algorithms. The master algorithm is used to implement two commonly used interval division algorithms: The Golden Section search and the Fibonacci Division.

6.1.1 General Interval Division

We now describe the ideas behind the interval division methods. For given $x_0, x_3 \in \mathbb{R}$, with $x_0 < x_3$, let $\mathbf{X} \triangleq [x_0, x_3]$. Suppose we want to minimize $f(\cdot)$ on $\mathbf{X}$, and suppose that $f\colon \mathbb{R} \to \mathbb{R}$ has a unique minimizer $x^* \in \mathbf{X}$. For some $s \in (0, 1)$, let

$$x_1 \triangleq x_0 + s\,(x_3 - x_0), \tag{6.1}$$

$$x_2 \triangleq x_1 + s\,(x_3 - x_1). \tag{6.2}$$

If $f(x_1) \leq f(x_2)$, then $x^* \in [x_0,\, x_2]$. Hence, we can eliminate the interval $(x_2,\, x_3]$ and restrict our search to $[x_0,\, x_2]$. Similarly, if $f(x_1) > f(x_2)$, then $x^* \in [x_1,\, x_3]$ and we can eliminate $[x_0,\, x_1)$. Thus, we reduced the initial inter-

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

val to a new interval that contains the minimizer x^*.

Let $i \in \mathbb{N}$ be the iteration number. We want to nest the sequence of intervals

$$[x_{0,(i+1)},\, x_{3,(i+1)}] \subset [x_{0,i},\, x_{3,i}], \qquad i \in \{0,\, 1,\, 2,\, \ldots\}, \tag{6.3}$$

such that we have to evaluate $f(\cdot)$ in each step at one new point only. To do so, we assign the new bounds of the interval such that either $[x_{0,(i+1)},\, x_{3,(i+1)}] = [x_{0,i},\, x_{2,i}]$, or $[x_{0,(i+1)},\, x_{3,(i+1)}] = [x_{1,i},\, x_{3,i}]$, depending on which interval has to be eliminated. By doing so, we have to evaluate only one new point in the interval. It remains to decide where to locate the new point. The Golden Section and Fibonacci Division differ in this decision.

6.1.2 Golden Section Interval Division

Suppose we have three points $x_0 < x_1 < x_3$ in $\mathbf{X} \subset \mathbb{R}$ such that for some $q \in (0, 1)$, to be determined later,

$$\frac{|x_0 - x_1|}{|x_0 - x_3|} = q. \tag{6.4a}$$

Hence,

$$\frac{|x_1 - x_3|}{|x_0 - x_3|} = 1 - q. \tag{6.4b}$$

Suppose that x_2 is located somewhere between x_1 and x_3 and define the ratio

$$w \triangleq \frac{|x_1 - x_2|}{|x_0 - x_3|}. \tag{6.5}$$

Depending on which interval is eliminated, the interval in the next iteration step will either be of length $(q + w)\,|x_0 - x_3|$, or $(1 - q)\,|x_0 - x_3|$. We select the location of x_2 such that the two intervals are of the same length. Hence,

$$q + w = 1 - q. \tag{6.6a}$$

Now, we determine the fraction q. Since we apply the process of interval division recursively, we know by scale similarity that

$$\frac{w}{1 - q} = q. \tag{6.6b}$$

Combining (6.6a) and (6.6b) leads to

$$q^2 - 3q + 1 = 0, \tag{6.7a}$$

with solutions

$$q_{1,2} = \frac{3 \pm \sqrt{5}}{2}. \tag{6.7b}$$

Since $q < 1$ by (6.4a), the solution of interest is

$$q = \frac{3 - \sqrt{5}}{2} \approx 0.382. \tag{6.7c}$$

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

The fractional distances $q \approx 0.382$ and $1 - q \approx 0.618$ correspond to the so-called *Golden Section*, which gives this algorithm its name.

Note that the interval is reduced in each step by the fraction $1 - q$, i.e., we have *linear convergence*. In the m-th iteration, we have

$$\begin{aligned} |x_{0,\,m} - x_{2,\,m}| &= |x_{1,\,m} - x_{3,\,m}| = |x_{0,\,(m+1)} - x_{3,\,(m+1)}| \\ &= (1-q)^{m+1}\,|x_{0,\,0} - x_{3,\,0}|. \end{aligned} \tag{6.8}$$

Hence, the required number of iterations, m, to reduce the initial interval of uncertainty $|x_{0,\,0} - x_{3,\,0}|$ to at least a fraction r, defined as

$$r \triangleq \frac{|x_{0,\,m} - x_{2,\,m}|}{|x_{0,\,0} - x_{3,\,0}|} = \frac{|x_{1,\,m} - x_{3,\,m}|}{|x_{0,\,0} - x_{3,\,0}|}, \tag{6.9}$$

is given by

$$m = \frac{\ln r}{\ln(1-q)} - 1. \tag{6.10}$$

6.1.3 Fibonacci Division

Another way to divide an interval such that we need one function evaluation per iteration can be constructed as follows: Given an initial interval $[x_{0,\,i}, x_{3,\,i}]$, $i = 0$, we divide it into three segments symmetrically around its midpoint. Let $d_{1,\,i} < d_{2,\,i} < d_{3,\,i}$ denote the distance of the segment endpoints, measured from $x_{0,\,i}$. Then we have by symmetry $d_{3,\,i} = d_{1,\,i} + d_{2,\,i}$. By the bracket elimination procedure explained above, we know that we are eliminating a segment of length $d_{1,\,i}$. Therefore, our new interval is of length $d_{3,\,(i+1)} = d_{2,\,i}$. By symmetry we also have $d_{3,\,(i+1)} = d_{1,\,(i+1)} + d_{2,\,(i+1)}$. Hence, if we construct our segment length such that $d_{3,\,(i+1)} = d_{1,\,(i+1)} + d_{2,\,(i+1)} = d_{2,\,i}$ we can reuse one known point. Such a construction can be done by using *Fibonacci* numbers, which are defined recursively by

$$F_0 \triangleq F_1 \triangleq 1, \tag{6.11a}$$

$$F_i \triangleq F_{i-1} + F_{i-2}, \qquad i \in \{2,\,3,\,\ldots\}. \tag{6.11b}$$

The first few numbers of the Fibonacci sequence are $\{1,\,1,\,2,\,3,\,5,\,8,\,13,\,21,\,\ldots\}$. The length of the intervals $d_{1,\,i}$ and $d_{2,\,i}$, respectively, are then given by

$$d_{1,\,i} = \frac{F_{m-i}}{F_{m-i+2}}, \quad d_{2,\,i} = \frac{F_{m-i+1}}{F_{m-i+2}}, \qquad i \in \{0,\,1,\,\ldots\,,\,m\}, \tag{6.12}$$

where $m > 0$ describes how many iterations will be done. Note that m must be known prior to the first interval division. Hence, the algorithm must be stopped after m iterations.

The reduction of the length of the uncertainty interval per iteration is given by

$$\frac{d_{3,\,(i+1)}}{d_{3,\,i}} = \frac{d_{2,\,i}}{d_{1,\,i} + d_{2,\,i}} = \frac{\frac{F_{m-i+1}}{F_{m-i+2}}}{\frac{F_{m-i}}{F_{m-i+2}} + \frac{F_{m-i+1}}{F_{m-i+2}}} = \frac{F_{m-i+1}}{F_{m-i+2}}. \tag{6.13}$$

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

After m iterations, we have

$$\begin{aligned}\frac{d_{3,\,m}}{d_{3,\,0}} &= \frac{d_{3,\,m}}{d_{3,\,(m-1)}}\,\frac{d_{3,\,(m-1)}}{d_{3,\,(m-2)}}\cdots\frac{d_{3,\,2}}{d_{3,\,1}}\,\frac{d_{3,\,1}}{d_{3,\,0}} \\ &= \frac{F_2}{F_3}\,\frac{F_3}{F_4}\cdots\frac{F_m}{F_{m+1}}\,\frac{F_{m+1}}{F_{m+2}} = \frac{2}{F_{m+2}}. \end{aligned} \tag{6.14}$$

The required number of iterations m to reduce the initial interval $d_{3,\,0}$ to at least a fraction r, defined by (6.9), can again be obtained by expansion from

$$\begin{aligned} r &= \frac{d_{2,\,m}}{d_{3,\,0}} = \frac{d_{3,\,(m+1)}}{d_{3,\,0}} = \frac{d_{3,\,(m+1)}}{d_{3,\,m}}\,\frac{d_{3,\,m}}{d_{3,\,(m-1)}}\cdots\frac{d_{3,\,2}}{d_{3,\,1}}\,\frac{d_{3,\,1}}{d_{3,\,0}} \\ &= \frac{F_1}{F_2}\,\frac{F_2}{F_3}\cdots\frac{F_m}{F_{m+1}}\,\frac{F_{m+1}}{F_{m+2}} = \frac{1}{F_{m+2}}. \end{aligned} \tag{6.15}$$

Hence, m is given by

$$m = \underset{m\in\mathbb{N}}{\arg\min}\left\{m \mid r \geq \frac{1}{F_{m+2}}\right\}. \tag{6.16}$$

6.1.4 Comparison of Efficiency

The Golden Section is more efficient than the Fibonacci Division. Comparing the reduction of the interval of uncertainty, $|x_{0,\,m} - x_{3,\,m}|$, in the limiting case for $m \to \infty$, we obtain

$$\lim_{m\to\infty}\frac{|x_{0,\,m} - x_{3,\,m}|_{GS}}{|x_{0,\,m} - x_{3,\,m}|_{F}} = \lim_{m\to\infty}\frac{F_{m+2}}{2}\,(1-q)^m = 0.95. \tag{6.17}$$

6.1.5 Master Algorithm for Interval Division

The following master algorithm explains the steps of the interval division algorithm.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Algorithm 6.1.1 (Model Interval Division Algorithm)

Data: x_0, x_3.
Procedure that returns r_i, defined as
$r_i \triangleq |x_{0,\,i} - x_{2,\,i}|/|x_{0,\,0} - x_{3,\,0}|$.

Step 0: *Initialize*
$\Delta x = x_3 - x_0$,
$x_2 = x_0 + r_1\,\Delta x$,
$x_1 = x_0 + r_2\,\Delta x$,
$f_1 = f(x_1)$, $f_2 = f(x_2)$, and
$i = 2$.

Step 1: *Iterate.*
Replace i by $i + 1$.
If $(f_2 < f_1)$
    Set $x_0 = x_1$, $x_1 = x_2$,
    $f_1 = f_2$,
    $x_2 = x_3 - r_i\,\Delta x$, and
    $f_2 = f(x_2)$.
else
    Set $x_3 = x_2$, $x_2 = x_1$,
    $f_2 = f_1$,
    $x_1 = x_0 + r_i\,\Delta x$,
    $f_1 = f(x_1)$.

Step 2: Stop or go to Step 1.

6.1.6 Keywords

For the Golden Section and the Fibonacci Division algorithm, the command file (see page 83) must contain only one continuous parameter.

To invoke the Golden Section or the Fibonacci Division algorithm, the `Algorithm` Section of the GenOpt command file must have following form:

```
Algorithm{
   Main              = GoldenSection | Fibonacci;
  [AbsDiffFunction   = Double;  |   // 0 < AbsDiffFunction
   IntervalReduction = Double;  ]   // 0 < IntervalReduction
}
```

The keywords have the following meaning

`Main` The name of the main algorithm.

The following two keywords are optional. If none of them is specified, then the algorithm stops after `MaxIte` function evaluations (i.e., after `MaxIte`−2 iterations), where `MaxIte` is specified in the section `OptimizationSettings`. If both of them are specified, an error occurs.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

`AbsDiffFunction` The absolute difference defined as

$$\Delta f \triangleq |\min\{f(x_0),\, f(x_3)\} - \min\{f(x_1),\, f(x_2)\}|. \tag{6.18}$$

If Δf is lower than `AbsDiffFunction`, the search stops successfully.
Note: Since the maximum number of interval reductions must be known for the initialization of the Fibonacci algorithm, this keyword can be used only for the Golden Section algorithm. It must not be specified for the Fibonacci algorithm.

`IntervalReduction` The required maximum fraction, r, of the end interval length relative to the initial interval length (see equation (6.9)).

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

7 Algorithms for Parametric Runs

The here described algorithms for parametric runs can be used to determine how sensitive a function is with respect to a change in the independent variables. They can also be used to do a parametric sweep of a function over a set of parameters. The algorithm described in Section 7.1 varies one parameter at a time while holding all other parameters fixed at the value specified by the keyword `Ini`. The algorithm described in Section 7.2, in contrast, constructs a mesh in the space of the independent parameters, and evaluates the objective function at each mesh point.

7.1 Parametric Runs by Single Variation

7.1.1 Algorithm Description

The `Parametric` algorithm allows doing parametric runs where one parameter at a time is varied and all other parameters are fixed at their initial values (specified by the keyword `Ini`).

Each parameter must have a lower and upper bound. For the logarithmic scale, the lower and upper bounds must be bigger than zero. To allow negative increments, the lower bound can be larger than the upper bound. The absolute value of the keyword `Step` defines in how many intervals each coordinate axis will be divided. If `Step` < 0, then the spacing is logarithmic; otherwise it is linear. Set `Step` $= 0$ to keep the parameter always fixed at the value specified by `Ini`.

This algorithm can also be used with discrete parameters. This allows, for example, using a string to specify a window construction.

The spacing is computed as follows: For simplicity, the explanation is done for one parameter. Let $l \triangleq$ `Min`, $u \triangleq$ `Max` and $m \triangleq |$`Step`$|$, where `Min`, `Max` and `Step` are specified in the command file.

If `Step` < 0, we compute, for $i \in \{0, \ldots, m\}$,

$$p = \frac{1}{m} \log \frac{u}{l}, \tag{7.1a}$$

$$x_i = l\, 10^{p\, i}. \tag{7.1b}$$

If `Step` > 0, we compute, for $i \in \{0, \ldots, m\}$,

$$x_i = l + \frac{i}{m}\,(u - l). \tag{7.1c}$$

Example 7.1.1 (Parametric run with logarithmic and linear spacing) Suppose the parameter specification is of the form

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

```
Vary{
   Parameter{ Name = x1; Ini = 5; Step = -2; Min = 10; Max = 1000; }
   Parameter{ Name = x2; Ini = 3; Step = 1;  Min = 2;  Max = 20;   }
}
```

and the cost function takes two arguments, $x_1, x_2 \in \mathbb{R}$. Then, the cost function will be evaluated at the points
$(x_1, x_2) \in \{(10, 3),\ (100, 3),\ (1000, 3),\ (5, 2),\ (5, 20)\}$. □

7.1.2 Keywords

For this algorithm, the command file (see page 83) can contain continuous and discrete parameters.

The `Parametric` algorithm is invoked by the following specification in the command file:

```
Algorithm{
   Main = Parametric;
   StopAtError = true | false;
}
```

The keywords have the following meaning:

`Main` The name of the main algorithm.

`StopAtError` If `true`, then the parametric run stops if a simulation error occurs. If `false`, then the parametric run does not stop if a simulation error occurs. The failed function evaluation will be assigned the function value zero. For information, an error message will be written to the user interface and the optimization log file.

7.2 Parametric Runs on a Mesh

7.2.1 Algorithm Description

In contrast to the algorithm `Parametric`, the algorithm `Mesh` spans a multi-dimensional grid in the space of the independent parameters, and it evaluates the objective function at each grid point.

Note that the number of function evaluations increases exponentially with the number of independent parameters. For example, a 5-dimensional grid with 2 intervals in each dimension requires $3^5 = 243$ function evaluations, whereas a 10-dimensional grid would require $3^{10} = 59049$ function evaluations.

The values that each parameter can take on are computed in the same way as for the algorithm `Parametric`. Therefore, the specification of a `Parameter` underlies the same constraints as for the algorithm `Parametric`, which is described above.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Example 7.2.1 (Parametric run on a mesh)
Suppose the parameter specification is of the form

```
Vary{
  Parameter{ Name = x1; Min = -10; Ini = 99; Max = 10; Step = 1; }
  Parameter{ Name = x2; Min = 1; Ini = 99; Max = 100; Step = -2; }
}
```

and the cost function takes two arguments, $x_1, x_2 \in \mathbb{R}$. Then, the cost function will be evaluated at the points
$(x_1, x_2) \in \{(-10, 1), (10, 1), (-10, 10), (10, 10), (-10, 100), (10, 100)\}$.

An alternative specification for x_2 that uses a discrete parameter and gives the same result is

```
  Parameter{ Name = x2; Ini = "1"; Values = "1, 10, 100"; }
```

□

7.2.2 Keywords

The `Mesh` algorithm is invoked by the following specification in the command file:

```
Algorithm{
   Main        = Mesh;
   StopAtError = true | false;
}
```

The keywords have the following meaning:

`Main` The name of the main algorithm.

`StopAtError` If `true`, then the parametric run stops if a simulation error occurs. If `false`, then the parametric run does not stop if a simulation error occurs. The failed function evaluation will be assigned the function value zero. For information, an error message will be written to the user interface and the optimization log file.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

8 Constraints

For some optimization problems it is necessary to impose constraints on the independent variables and/or the dependent variables, as the following example shows.

Example 8.0.2 Suppose we want to minimize the heating energy of a building, and suppose that the normalized mass flow $\dot{m}$ of the heating system is an independent variable, with constraints $0 \leq \dot{m} \leq 1$. Without using constraints, the minimum energy consumption would be achieved for $\dot{m} = 0$, since then the heating system is switched off. To solve this problem, we can impose a constraint on a dependent variable. One possibility is to add a "penalty" term to the energy consumption. This could be such that every time a thermal comfort criterion (which is a dependent variable) is violated, a large positive number is added to the energy consumption. Thus, if $\mathrm{ppd}(x)$, with $\mathrm{ppd}\colon \mathbb{R}^n \to \mathbb{R}$, denotes the predicted percent of dissatisfied people (in percentage), and if we require that $\mathrm{ppd}(x) \leq 10\%$, we could use the inequality constraint $g(x) \triangleq \mathrm{ppd}(x) - 10 \leq 0$. □

In Section 8.1.1, the method that is used in GenOpt to implement box constraints is described. In Section 8.2, penalty and barrier methods that can be used to implement constraints on dependent variables are described. They involve reformulating the cost function and, hence, are problem specific and have to be implemented by the user.

8.1 Constraints on Independent Variables

8.1.1 Box Constraints

Box constraints are constant inequality constraints that define a feasible set as

$$\mathbf{X} \triangleq \left\{x \in \mathbb{R}^n \mid l^i \leq x^i \leq u^i,\ i \in \{1, \ldots, n\}\right\}, \tag{8.1}$$

where $-\infty \leq l^i < u^i \leq \infty$ for $i \in \{1, \ldots, n\}$.

In GenOpt, box constraints are either implemented directly in the optimization algorithm by setting $f(x) = \infty$ for unfeasible iterates, or, for some algorithms, the independent variable $x \in \mathbf{X}$ is transformed to a new unconstrained variable which we will denote in this section by $t \in \mathbb{R}^n$.

Instead of optimizing the constrained variable $x \in \mathbf{X}$, we optimize with respect to the unconstrained variable $t \in \mathbb{R}^n$. The transformation ensures that all variables stay feasible during the iteration process. In GenOpt, the following transformations are used:

If $l^i \leq x^i$, for some $i \in \{1, \ldots, n\}$,

$$t^i = \sqrt{x^i - l^i}, \tag{8.2a}$$

$$x^i = l^i + (t^i)^2. \tag{8.2b}$$

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

If $l^i \leq x^i \leq u^i$, for some $i \in \{1, \ldots, n\}$,

$$t^i = \arcsin\left(\sqrt{\frac{x^i - l^i}{u^i - l^i}}\right), \tag{8.2c}$$

$$x^i = l^i + (u^i - l^i)\,\sin^2 t^i. \tag{8.2d}$$

If $x^i \leq u^i$, for some $i \in \{1, \ldots, n\}$,

$$t^i = \sqrt{u^i - x^i}, \tag{8.2e}$$

$$x^i = u^i - (t^i)^2. \tag{8.2f}$$

8.1.2 Coupled Linear Constraints

In some cases the constraints have to be formulated in terms of a linear system of equations of the form

$$A\,x = b, \tag{8.3}$$

where $A \in \mathbb{R}^m \times \mathbb{R}^n$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $\mathrm{rank}(A) = m$.

There are various algorithms that take this kind of restriction into account. However, such restrictions are rare in building simulation and thus not implemented in GenOpt. If there is a need to impose such restrictions, they can be included by adding an appropriate optimization algorithm and retrieving the coefficients by using the methods offered in GenOpt's class `Optimizer`.

8.2 Constraints on Dependent Variables

We now discuss the situation where the constraints are non-linear and defined by

$$g(x) \leq 0, \tag{8.4}$$

where $g\colon \mathbb{R}^n \to \mathbb{R}^m$ is once continuously differentiable. (8.4) also allows formulating equality constraints of the form

$$h(x) = 0, \tag{8.5}$$

for $h\colon \mathbb{R}^n \to \mathbb{R}^m$, which can be implemented by using penalty functions. In example, one can define $g^i(x) \triangleq h^i(x)^2$ for $i \in \{1, \ldots, m\}$. Then, since $g^i(\cdot)$ is non-negative, the only feasible value is $g(\cdot) = 0$. Thus, we will only discuss the case of inequality constraints of the form (8.4).

Such a constraint can be taken into account by adding *penalty* or *barrier* functions to the cost function, which are multiplied by a positive weighting factor μ that is monotonically increased (for penalty functions) or monotonically decreased to zero (for barrier functions).

We now discuss the implementation of barrier and penalty functions.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

8.2.1 Barrier Functions

Barrier functions impose a punishment if the dependent variable gets close to the boundary of the feasible region. The closer the variable is to the boundary, the higher the value of the barrier function becomes.
To implement a barrier function for $g(x) \leq 0$, where $g\colon \mathbb{R}^n \to \mathbb{R}^m$ is a continuously differentiable function whose elements are strictly monotone increasing, the cost function $f\colon \mathbb{R}^n \to \mathbb{R}$ can be modified to

$$\widetilde{f}(x,\mu) \triangleq f(x) + \mu \frac{1}{\sum_{i=1}^{m} g^i(x)} \tag{8.6}$$

where $\widetilde{f}\colon \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$. The optimization algorithm is then applied to the new function $\widetilde{f}(x,\mu)$. Note that (8.6) requires that x is in the interior of the feasible set[1].

A drawback of barrier functions is that the boundary of the feasible set can not be reached. By selecting the weighting factors small, one can get close to the boundary. However, too small a weighting factor can cause the cost function to be ill-conditioned, which can cause problems for the optimization algorithm.

Moreover, if the variation of the iterates between successive iterations is too big, then the feasible boundary can be crossed. Such a behavior must be prevented by the optimization algorithm, which can produce additional problems.

For barrier functions, one can start with a moderately large weighting factor μ and let μ tend to zero during the optimization process. That is, one constructs a sequence

$$\mu_0 > \ldots > \mu_i > \mu_{i+1} > \ldots > 0. \tag{8.7}$$

Section 8.2.3 shows how μ_i can be computed in the coarse of the optimization.

Barrier functions do not allow formulating equality constraints of the form (8.5).

8.2.2 Penalty Functions

In contrast to barrier functions, penalty functions allow crossing the boundary of the feasible set, and they allow implementation of equality constraints of the form (8.5). Penalty functions add a positive term to the cost function if a constraint is violated.
To implement a penalty function for $g(x) \leq 0$, where $g\colon \mathbb{R}^n \to \mathbb{R}^m$ is once continuously differentiable and each element is strictly monotone decreasing, the cost function $f\colon \mathbb{R}^n \to \mathbb{R}$ can be modified to

$$\widetilde{f}(x,\mu) \triangleq f(x) + \mu \sum_{i=1}^{m} \max(0, g^i(x))^2, \tag{8.8}$$

where $\widetilde{f}\colon \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ is once continuously differentiable in x. The optimization algorithm is then applied to the new function $\widetilde{f}(x,\mu)$.

[1] I.e., x satisfies the strict inequality $g(x) > 0$.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

As for the barrier method, selecting the weighting factor μ is not trivial. Too small a value for μ produces too big a violation of the constraint. Hence, the boundary of the feasible set can be exceeded by an unacceptable amount. Too large a value of μ can lead to ill-conditioning of the cost function, which can cause numerical problems.

The weighting factors have to satisfy

$$0 < \mu_0 < \ldots < \mu_i < \mu_{i+1} < \ldots, \tag{8.9}$$

with $\mu_i \to \infty$, as $i \to \infty$. See Section 8.2.3 for how to adjust μ_i.

8.2.3 Implementation of Barrier and Penalty Functions

We now discuss how the weighting factors μ_i can be adjusted. For $i \in \mathbb{N}$, let $x^*(\mu_i)$ be defined as the solution

$$x^*(\mu_i) \triangleq \arg\min_{x \in \mathbf{X}} \widetilde{f}(x, \mu_i), \tag{8.10}$$

where $\widetilde{f}(x, \mu_i)$ is as in (8.6) or (8.8), respectively. Then, we initialize $i = 0$, select an initial value $\mu_0 > 0$ and compute $x^*(\mu_0)$. Next, we select a μ_{i+1} such that it satisfies (8.7) (for barrier functions) or (8.9) (for penalty functions), and compute $x^*(\mu_{i+1})$, using the initial iterate $x^*(\mu_i)$, and increase the counter i to $i + 1$. This procedure is repeated until μ_i is sufficiently close to zero (for barrier functions) or sufficiently large (for penalty functions).

To recompute the weighting factors μ_i, users can request GenOpt to write a counter to the simulation input file, and then compute μ_i as a function of this counter. The value of this counter can be retrieved by setting the keyword `WriteStepNumber` in the optimization command file to `true`, and specifying the string `%stepNumber%` in the simulation input template file. GenOpt will replace the string `%stepNumber%` with the current counter value when it writes the simulation input file. The counter starts with the value 1 and its increment is 1.

Users who implement their own optimization algorithm in GenOpt can call the method `increaseStepNumber(...)` in the class `Optimizer` to increase the counter. If the keyword `WriteStepNumber` in the optimization command file is set to `true`, the method calls the simulation to evaluate the cost function for the new value of this counter. If `WriteStepNumber` is `false`, no new function evaluation is performed by this method since the cost function does not depend on this counter.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Input Files

initialization:	Specification of file location (input files, output files, log files, etc.)
command:	Specification of parameter names, initial values, upper/lower bounds, optimization algorithm, etc.
configuration:	Configuration of simulation program (error indicators, start command, etc.)
simulation input template:	Templates of simulation input files

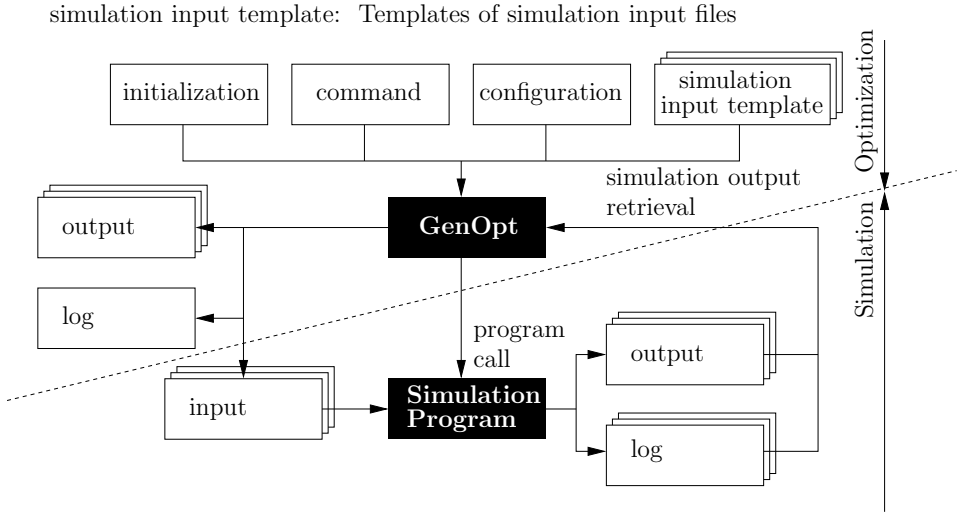


Figure 9.1: Interface between GenOpt and the simulation program that evaluates the cost function.

9 Program

GenOpt is divided into a kernel part and an optimization part. The kernel reads the input files, calls the simulation program, stores the results, writes output files, etc. The optimization part contains the optimization algorithms. It also contains classes of mathematical functions such as those used in linear algebra.

Since there is a variety of simulation programs and optimization algorithms, GenOpt has a simulation program interface and an optimization algorithm interface. The simulation program interface allows using any simulation software to evaluate the cost function (see below for the requirements on the simulation program), and allows implementing new optimization algorithms with little effort.

9.1 Interface to the Simulation Program

Text files are used to exchange data with the simulation program and to specify how to start the simulation program. This makes it possible to couple

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

any simulation program to GenOpt without requiring code adaptation on either the GenOpt side or the simulation program side. The simulation program must satisfy the following requirements:

1. The simulation program must read its input from one or more text files, must write the value of the cost function to a text file, and must write error messages to a text file.
2. It must be able to start the simulation program by a command and the simulation program must terminate automatically. This means that the user does not have to open the input file manually and shut down the simulation program once the simulation is finished.

The simulation program may be a commercially available program or one written by the user.

9.2 Interface to the Optimization Algorithm

The large variety of optimization algorithms led to the development of an open interface that allows easy implementation of optimization algorithms. Users can implement their own algorithms and add them to the library of available optimization algorithms without having to adapt and recompile GenOpt. To implement a new optimization algorithm, the optimization algorithm must be written according to the guidelines of Section 9.4. Thus, GenOpt can not only be used to do optimization with built-in algorithms, but it can also be used as a framework for developing, testing and comparing optimization algorithms.

Fig. 9.2 shows GenOpt's program structure. The class `Optimizer` is the superclass of each optimization algorithm. It offers all the functions required for retrieval of parameters that specify the optimization settings, performing the evaluation of the cost function and reporting results. For a listing of its methods, see `http://SimulationResearch.lbl.gov` or the Javadoc code documentation that comes with GenOpt's installation.

9.3 Package `genopt.algorithm`

The Java package `genopt.algorithm` consists of all classes that contain mathematical formulas that are used by the optimization algorithm. The following packages belong to `genopt.algorithm`.

`genopt.algorithm` This package contains all optimization algorithms. The abstract class `Optimizer`, which must be inherited by each optimization algorithm, is part of this package.

`genopt.algorithm.util.gps` contains a model Generalized Pattern Search optimization algorithm.

`genopt.algorithm.util.linesearch` contains classes for doing a line search along a given direction.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

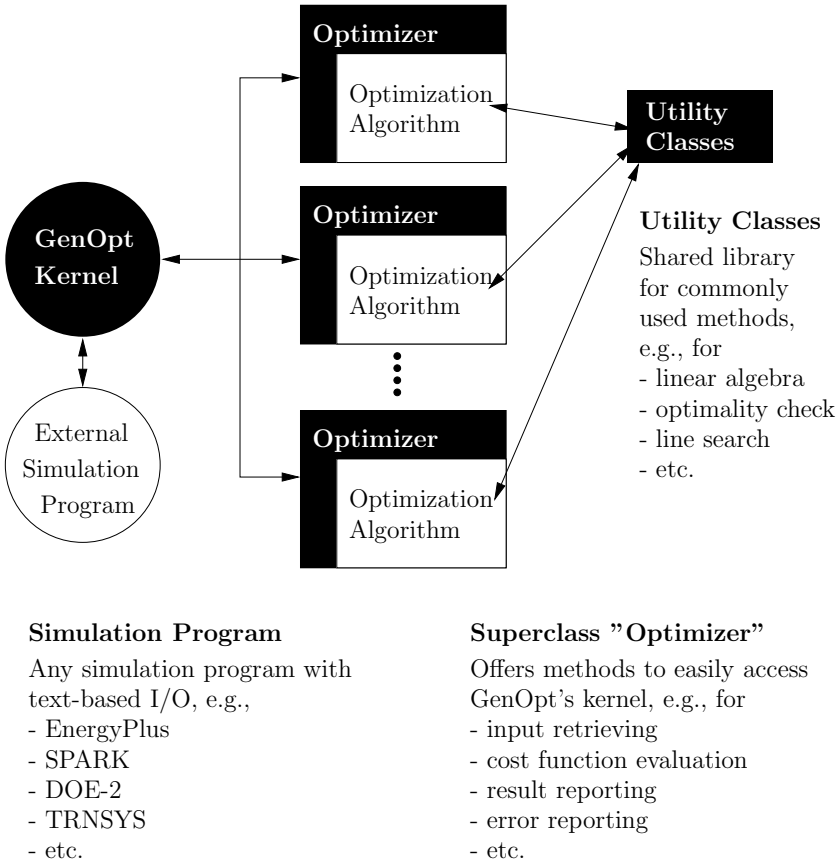


Figure 9.2: Implementation of optimization algorithms into GenOpt.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

`genopt.algorithm.util.math` contains classes for mathematical operations.

`genopt.algorithm.util.optimality` contains classes that can be used to check whether a variable value is at a minimum point or not.

`genopt.algorithm.util.pso` contains a model Particle Swarm Optimization algorithm.

These packages are documented in the Javadoc source code documentation that comes with GenOpt's.

9.4 Implementing a New Optimization Algorithm

To implement a new optimization algorithm, you must write a Java class that has the syntax shown in Fig. 9.3. The class must use the methods of the abstract class `Optimizer` to evaluate the cost function and to report the optimization steps. The methods of the `Optimizer` class are documented in the Javadoc source code documentation.

Follow these steps to implement and use your own optimization algorithm:

1. Put the byte-code (*ClassName*`.class`) in the directory `genopt/algorithm`.
2. Set the value of the keyword `Main` in the `Algorithm` section of the optimization command file to the name of the optimization class (without file extension).
3. Add any further keywords that the algorithm requires to the `Algorithm` section. The keywords must be located *after* the entry `Main` of the optimization command file. The keywords must be in the same sequence as they are called in the optimization code.
4. Call the method `Optimizer.report(final Point, final boolean)` after evaluating the cost function. Otherwise, the result will not be reported.
5. Call either the method `Optimizer.increaseStepNumber()` or the method `Optimizer.increaseStepNumber(final Point)` after the optimization algorithm converged to some point. These methods increase a counter that can be used to add penalty or barrier functions to the cost function. In particular, the methods `Optimizer.increaseStepNumber()` and `Optimizer.increaseStepNumber(finalPoint)` increase the variable `stepNumber` (see Section 8) by one.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

```
package genopt.algorithm;

import genopt.GenOpt;
import genopt.lang.OptimizerException;
import genopt.io.InputFormatException;

public class ClassName extends Optimizer{

   public ClassName (GenOpt genOptData)
      throws InputFormatException, OptimizerException,
             IOException, Exception
   {
      // set the mode to specify whether the
      // default transformations for the box
      // constraints should be used or not
      int constraintMode = xxxx;
      super(genOptData, constraintMode);

      // remaining code of the constructor
   }

   public int run() throws OptimizerException, IOException
   {
      // the code of the optimization algorithm
   }

   // add any further methods and data members
}
```

Figure 9.3: Code snippet that specifies how to implement an optimization algorithm.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

10 Installing and Running GenOpt

10.1 System Requirements

To run GenOpt and the GenOpt installation program, a Java 2 v1.5.0 runtime environment is required. Java may be obtained from `http://java.sun.com`. GenOpt should run on any operating system that can run Java applications.

10.2 Installing and uninstalling GenOpt

To install GenOpt, download the installation program `genopt-install.jar` from `http://SimulationResearch.lbl.gov/GO`. Then, either double-click on the file `genopt-install.jar`[1] or open a command shell, change to the directory that contains `genopt-install.jar` and type

```
java -jar genopt-install.jar
```

No environment variables need to be set to run GenOpt. (This is new since GenOpt 2.1.0.)

To uninstall GenOpt, delete the directory in which GenOpt was installed.

10.3 Running GenOpt

10.3.1 Running GenOpt from the file explorer

To run GenOpt from the file explorer, double-click on the file `genopt.jar`.[1] This will start the graphical user interface. From the graphical user interface, select `File`, `Start...` and select a GenOpt initialization file.

10.3.2 Running GenOpt from the command line

GenOpt can also be run as a console application, either with or without the graphical user interface. To run GenOpt as a console application with the graphical user interface, open a shell, change to the directory that contains `genopt.jar` and type

```
java -jar genopt.jar [initializationFile]
```

where `[initializationFile]` is an optional argument that can be replaced with the GenOpt initialization file (see example below). To start GenOpt without the graphical user interface, type

```
java -classpath genopt.jar genopt.GenOpt [initializationFile]
```

[1] Depending on your Java installation, the file extension `jar` may not be associated with Java. In this situation, please consult the instructions of your operating system for how to associate file extensions with programs.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

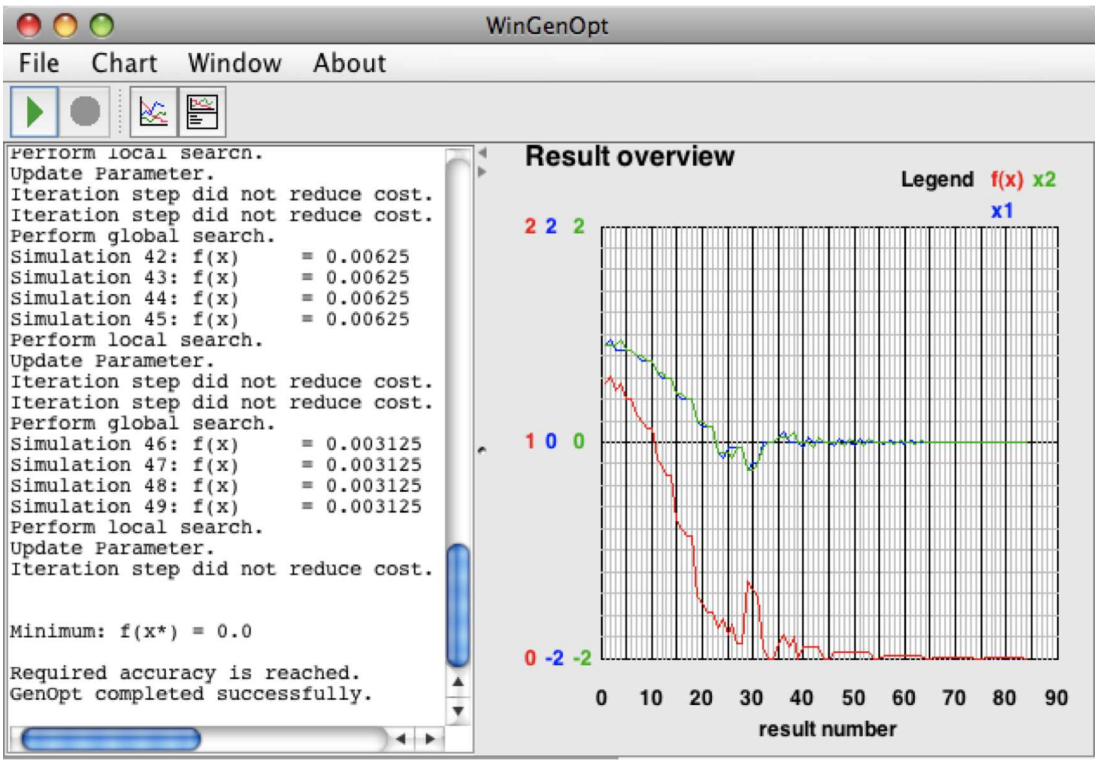


Figure 10.1: Output of GenOpt on Mac OS X for the example file in the directory `example/quad/GPSHookeJeeves`.

For instance, to run the example file provided with GenOpt that minimizes a quadratic function using the Hooke-Jeeves algorithm, type on Mac OS X

```
java -jar genopt.jar example/quad/GPSHookeJeeves/optMacOSX.ini
```

on Linux

```
java -jar genopt.jar example/quad/GPSHookeJeeves/optLinux.ini
```

and on Windows

```
java -jar genopt.jar example\quad\GPSHookeJeeves\optWinXP.ini
```

This should produce the window shown in Fig. 10.1.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

11 Setting Up an Optimization Problem

We will now discuss how to set up an optimization problem.

First, define a cost function. The cost function is the function that needs to be minimized. It must be evaluated by an external simulation program that satisfies the requirements listed on page 69. To maximize a cost function, change the sign of the cost function to turn the maximization problem into a minimization problem.

Next, specify possible constraints on the independent variables or on dependent variables (dependent variables are values that are computed in the simulation program). To do so, use the default scheme for box constraints on the independent variables or add penalty or barrier functions to the cost function as described in Chapter 8.

Next, make sure that the simulation program writes the cost function value to the simulation output file. *It is important that the cost function value is written to the output file without truncating any digits* (see Section 11.4). For example, if the cost function is computed by a Fortran program in double precision, it is recommended to use the `E24.16` format in the write statement.

In the simulation output file, the cost function value must be indicated by a string that stands in front of the cost function value (see page 80).

Then, specify the files described in Section 11.1 and, if required, implement pre- and post-processing, as described in Section 11.3.

11.1 File Specification

This section defines the file syntax for GenOpt. The directory `example` of the GenOpt installation contains several examples.

The following notation will be used to explain the syntax:

1. Text that is part of the file is written in `fixed width fonts`.
2. `|` stands for possible entries. Only one of the entries that are separated by `|` is allowed.
3. `[ ]` indicates optional values.
4. The file syntax follows the Java convention. Hence,
   (a) `//` indicates a comment on a single line,
   (b) `/*` and `*/` enclose a comment,
   (c) the equal sign, `=`, assigns values,
   (d) a statement has to be terminated by a semi-colon, `;`,
   (e) curly braces, `{ }`, enclose a whole section of statements, and

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

(f) the syntax is case sensitive.

The following basic types are used:

`String`	Any sequence of characters. If the sequence contains a blank character, it has to be enclosed in apostrophes (`"`). If there are apostrophes within quoted text, they must be specified by a leading backslash (i.e., `\"`). Similarly, a backslash must be preceded by another backslash (i.e., `"c:\\go_prg"`).
`StringReference`	Any name of a variable that appears in the same section.
`Integer`	Any integer value.
`Double`	Any double value (including integer).
`Boolean`	Either `true` or `false`

The syntax of the GenOpt files is structured into sections of parameters that belong to the same object. The sections have the form

```
ObjectKeyWord { Object }
```

where `Object` can either be another `ObjectKeyWord` or an assignment of the form

```
Parameter = Value ;
```

Some variables can be referenced. References have to be written in the form

```
Parameter = ObjectKeyWord1.ObjectKeyWord2.Value ;
```

where `ObjectKeyWord1` refers to the root of the object hierarchy as specified in the corresponding file.

11.1.1 Initialization File

The initialization file specifies

1. where the *files* of the current optimization problems are located,
2. which simulation files the user likes to have saved for later inspection,
3. what additional strings have to be passed to the command that starts the simulation (such as the name of the simulation input file),
4. what number in the simulation output file is a cost function value,
5. whether and if so, how, the cost function value(s) have to be post-processed, and
6. which simulation program is being used.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

The sections must be specified in the order shown below. The order of the keywords in each section is arbitrary, as long as the numbers that follow some keywords (such as `File1`) are in increasing order.

The initialization file syntax is

```
Simulation {
  Files {
    Template {
      File1 = String | StringReference;
    [ Path1 = String | StringReference; ]
      [ File2 = String | StringReference;
      [ Path2 = String | StringReference; ]
        [ ... ] ]
    }
    Input { // the number of input file must be equal to
            // the number of template files
      File1     = String | StringReference;
    [ Path1     = String | StringReference; ]
    [ SavePath1 = String | StringReference; ]
      [ File2     = String | StringReference;
      [ Path2     = String | StringReference; ]
      [ SavePath2 = String | StringReference; ]
        [ ... ] ]
    }
    Log {
```

The `Log` *section has the same syntax as the* `Input` *section.*

```
    }
    Output {
```

The `Output` *section has the same syntax as the* `Input` *section.*

```
    }
    Configuration {
      File1 = String | StringReference;
    [ Path1 = String | StringReference; ]
    }
  } // end of section Simulation.Files
  [CallParameter {
    [Prefix = String | StringReference;]
    [Suffix = String | StringReference;]
  }]
  [ObjectiveFunctionLocation {
    Name1       = String;
    Delimiter1  = String | StringReference;  |  Function1  = String;

    [ Name2       = String;
      Delimiter2  = String | StringReference;  |  Function2  = String;
```

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

```
    [ ... ] ]

  }]
} // end of section Simulation
Optimization {
  Files {
    Command {
      File1 = String | StringReference;
    [ Path1 = String | StringReference; ]
    }
  }
} // end of section Optimization
```

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

The sections have the following meaning:

`Simulation.Files.Template` GenOpt writes the value of the independent variables to the simulation input files. To do so, GenOpt reads the simulation input *template* files, replaces each occurrence of `%variableName%` by the numerical value of the corresponding variable, and the resulting file contents are written as the simulation input files. The string `%variableName%` refers to the name of the variable as specified by the entry `Name` in the optimization command file on page 83.

The independent variables can be written to several simulation input files if required. To do so, specify as many `Filei` and `Pathi` assignments as necessary (where `i` stands for a one-based counter of the file and path name). Note that there must obviously be the same number of files and paths in the `Input` section that follows this section.

If there are multiple simulation input template files, each file will be written to the simulation input file whose keyword ends with the same number.

The following rules are imposed:

1. Each variable name specified in the optimization command file *must* occur in at least one simulation input template file or in at least one function that is specified in the section `ObjectiveFunctionLocation` below.
2. Multiple occurrences of the same variable name are allowed in the same file and in the same function specification (as specified by the keyword `Functioni`, `i` $= 1, 2, \ldots$).
3. If the value `WriteStepNumber` in the section `OptimizationSettings` of the optimization command file is set to `true`, then rule 1 and 2 apply also to `%stepNumber%`. If `WriteStepNumber` is set to `false`, then `%stepNumber%` can occur, but it will be ignored.

`Simulation.Files.Input` The simulation input file is generated by GenOpt based on the current parameter set and the corresponding simulation input *template* file, as explained in the previous paragraph. Obviously, the number of simulation input files must be equal to the number of simulation input template files.

The section `Input` has an optional keyword, called `SavePath`. If `SavePath` is specified, then the corresponding input file will after each simulation be copied into the directory specified by `SavePath`. The copied file will have the same name, but with the simulation number added as prefix.

`Simulation.Files.Log` GenOpt scans the simulation log file for error messages. The optimization terminates if any of the strings specified by the

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

variable `ErrorMessage` in the `SimulationError` section of the GenOpt configuration file is found. At least one simulation log file must be specified.

The section `Log` also has the optional keyword `SavePath`. It has the same functionality as explained in the previous section.

`Simulation.Files.Output` GenOpt reads the cost function value from this file. GenOpt assumes that the value that is written after the *last* occurrence of the string specified by `Delimiteri` (`i` $= 1, 2, \ldots$) in the section `ObjectiveFunctionLocation` is the cost function value. The number of cost function values is arbitrary (but at least one must be specified). The optimization algorithms minimize the first cost function value. The other values can be used for post-processing of the simulation output. They will also be reported to the output files and the online chart.

GenOpt searches for the cost function value as follows:

1. After the first simulation, GenOpt searches for the first cost function value in the first output file. The number that occurs after the *last* occurrence of the string specified by the variable `Delimiteri` (`i` $= 1, 2, \ldots$) in the section `ObjectiveFunctionLocation` is assumed to be the cost function value. If the first output file does not contain the first cost function value, then GenOpt reads the second output file (if present) and so on until the last output file is read. If GenOpt cannot find the cost function value in any of the output files or function definitions, it will terminate with an error. The same procedure is repeated with the second cost function value (if present) until all cost function values have been found.
2. In the following iterations, GenOpt will only read the file(s) where it found the cost function value(s) after the first simulation. The files that did not contain a cost function value after the first simulation will not be read anymore.

This section also contains the optional keyword `SavePath`. If this keyword is specified, then GenOpt copies the output file. This is particularly useful for doing parametric runs.

`Simulation.Files.Configuration` The entries in this section specify the simulation configuration file, which contains information that is related to the simulation program only, but not related to the optimization problem. The simulation configuration file is explained below.

`Simulation.CallParameter` Here, a prefix and suffix for the command that starts the simulation program can be added. With these entries, any additional information, such as the name of the weather file, can be passed to the simulation program. To do so, one has to refer to either of these entries in the argument of the keyword `Command` (see page 82).

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

`Simulation.ObjectiveFunctionLocation` This section specifies where the cost function values can be found in the simulation output files, and possibly how these values have to be post-processed before they will be passed to the optimization algorithm.

GenOpt reads the value after the *last* occurrence of `Delimiteri` (`i` $= 1, 2, \ldots$) as the cost function value. The value of `Namei` is used to label the results in the output reports.

Alternatively to the entry `Delimiteri`, an entry `Functioni` can be specified to define how the cost function values should be post-processed. See page 89 for an example.

For convenience, the section `ObjectiveFunctionLocation` can optionally be specified in the initialization file, but its specification is required in the configuration file. If this section is specified in both files, then the specification in the initialization file will be used.

Specifying the section `ObjectiveFunctionLocation` in the initialization file is of interest if a simulation program is used for different problems that require different values of this section. Then, the same (simulation program specific) configuration file can be used for all runs and the different settings can be specified in the (project dependent) initialization file rather than in the configuration file.

`Optimization.Files.Command` This section specifies where the optimization command file is located. This file contains the mathematical information of the optimization. See page 83 for a description of this file.

11.1.2 Configuration File

The configuration file contains information related only to the simulation program used and not to the optimization problem. Hence, it has to be written only once for each simulation program and operating system. We recommend to put this file in the directory `cfg` so that it can be used for different optimization projects. Some configuration files are provided with the GenOpt installation.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

The syntax is specified by

```
// Error messages of the simulation program.
SimulationError{
   ErrorMessage = String;
  [ErrorMessage = String;
  [ ... ] ]
}

// Number format for writing simulation input files.
IO{
   NumberFormat = Float | Double;
}

// Specifying how to start the simulation program.
SimulationStart{
   Command = String;
   WriteInputFileExtension = Boolean;
}

// Specifying the location of the
// cost function value in the simulation output file
ObjectiveFunctionLocation{
    Name1       = String;
    Delimiter1  = String | StringReference;  |  Function1  = String;

    [ Name2       = String;
      Delimiter2  = String | StringReference;  |  Function2  = String;
    [ ... ] ]
}
```

The entries have the following meaning:

`SimulationError` The error messages that might be written by the simulation program must be assigned to the keyword `ErrorMessage` so that GenOpt can check whether the simulation has completed successfully. At least one entry for `ErrorMessage` must be given.

`IO` The keyword `NumberFormat` specifies in what format the independent parameters will be written to the simulation input file. The setting `Double` is recommended, unless the simulation program cannot read this number format.

`SimulationStart` The keyword `Command` specifies what string must be used to start the simulation program. It is important that this command waits until the simulation terminates (see the directory `cfg` for examples). The value of the variable `Command` is treated in a special way: Any value of the optimization initialization file can be automatically copied into the

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

value of `Command`. To do so, surround the reference to the corresponding keyword with percent signs. For example, a reference to the keyword `Prefix` of the initialization file looks like

```
%Simulation.CallParameter.Prefix%
```

By setting `WriteInputFileExtension` to `false`, the value of the keyword `Simulation.Input.Filei` (where `i` stands for `1`, `2`, `3`) is copied into `Command`, and the file extension is removed.

`ObjectiveFunctionLocation` Note that this section can also be specified in the initialization file. The section in this file is ignored if this section is also specified in the configuration file. See page 81 for a description.

11.1.3 Command File

The command file specifies optimization-related settings such as the independent parameters, the stopping criteria and the optimization algorithm being used. The sequence of the entries in all sections of the command file is arbitrary.

There are two different types of independent parameters, *continuous parameters* and *discrete parameters*. Continuous parameters can take on any values, possibly constrained by a minimum and maximum value. Discrete parameters can take on only user-specified discrete values, to be specified in this file.

Some algorithms require all parameters to be continuous, or all parameters to be discrete, or allow both continuous and discrete parameters. Please refer to the algorithm section on page 15-63.

a) Specification of a Continuous Parameter

The structure for a continuous parameter is

```
// Settings for a continuous parameter
Parameter{
   Name = String;
   Ini  = Double;
   Step = Double;
   [ Min  = Double | SMALL; ]
   [ Max  = Double | BIG; ]
   [ Type = CONTINUOUS; ]
}
```

The entries are:

`Name` The name of the independent variable. GenOpt searches the simulation input template files for this string – surrounded by percent signs – and replaces each occurrence by its numerical value before it writes the simulation input files.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

`Ini` Initial value of the parameter.

`Step` Step size of the parameter. How this variable is used depends on the optimization algorithm being used. See the optimization algorithm descriptions for details.

`Min` Lower bound of the parameter. If the keyword is omitted or set to `SMALL`, the parameter has no lower bound.

`Max` Upper bound of the parameter. If the keyword is omitted or set to `BIG`, the parameter has no upper bound.

`Type` Optional keyword that specifies that this parameter is continuous. By default, if neither `Type` nor `Values` (see below) are specified, then the parameter is considered to be continuous and the `Parameter` section must have the above format.

b) Specification of a Discrete Parameter

For discrete parameters you need to specify the set of admissible values. Alternatively, if a parameter is spaced either linearly or logarithmically, specify the minimum and maximum value of the parameter and the number of intervals.

First, we list the entry for the case of specifying the set of admissible values:

```
// Settings for a discrete parameter
Parameter{
   Name   = String;
   Ini    = Integer;
   Values = String;
 [ Type   = SET; ]
}
```

The entries are:

`Name` *As for the continuous parameter above.*

`Ini` 1-based index of the initial value. For example, if `Values` specifies three admissible values, then `Ini` can be either `1`, `2`, or `3`.

`Values` Set of admissible values. The entry must be of the form

```
Values = "value1, value2, value3";
```

i.e., the values are separated by a comma, and the list is enclosed in apostrophes (`"`). For `value1`, `value2`, etc., numbers and strings are allowed.
If all entries of `Values` are numbers, then the result reports contain the actual values of this entry. Otherwise, the result reports contain the index of this value, i.e., `1` corresponds to `value1`, `2` corresponds to `value2`, etc.

`Type` Optional keyword that specifies that this parameter is discrete. By default, if the entry `Values` is specified, a parameter is considered to be discrete, and the `Parameter` section must have the above format.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

To obtain linear or logarithmic spacing between a minimum and maximum value, the `Parameter` section can be specified as

```
// Settings for a discrete parameter, linearly or logarithmically spaced
Parameter{
   Name = String;
   Ini  = Integer;
   Type = SET;
   Min  = Double;
   Max  = Double;
   Step = Integer;
}
```

`Name` *As for the continuous parameter above.*

`Ini` 1-based index of the initial value. For example, if `Step` is set to +2 or to −2, then `Ini` can be set to any integer between 1 and 3.

`Type` This variable must be equal to `SET`.

`Min` Minimum value of the spacing.

`Max` Maximum value of the spacing.

`Step` Number of intervals. If `Step` < 0, then the spacing is logarithmic, otherwise it is linear. Set `Step` $= 0$ to keep the parameter always fixed on its minimum value.

The linear or logarithmic spacing is computed using (7.1) on page 61.

c) Specification of Input Function Objects

The specification of input function objects in optional. If any input function object is specified, then its name must appear either in another input function object, in a simulation input template file, or in an output function object. Otherwise, GenOpt terminates with an error message. See Section 11.3 on page 89 for an explanation of input and output function objects.

The syntax for input function objects is

```
// Input function objects entry
Function{
   Name     = String;
   Function = String;
}
```

The entries are

`Name` A unique name that is not used for any other input function object and for any other independent parameter.

`Function` A function object (see Section 11.3 on page 89). The string must be enclosed by apostrophes (`"`).

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

d) Structure of the Command File

Using above structures of the `Parameter` section, the command file has the structure

```
// Settings of the independent parameters
Vary{
   // Parameter entry
     List any of the Parameter  sections as described
     in the Sections 11.1.3.a) and 11.1.3.b).

   // Input function object
     List any of the Function  sections as described
     in the Section 11.1.3.c).
}

// General settings for the optimization process
OptimizationSettings{
   MaxIte           = Integer;
   WriteStepNumber  = Boolean;
 [ MaxEqualResults  = Integer; ]
 [ UnitsOfExecution = Integer; ]
}

// Specification of the optimization algorithm
Algorithm{
   Main = String;
   ... // any other entries that are required
       // by the optimization algorithm
}
```

The different sections are:

`Vary` This section contains the definition of the independent parameter and the input function objects. See Sections 11.1.3.a), 11.1.3.b), and 11.1.3.c) for possible entries.

`OptimizationSettings` This section specifies general settings of the optimization. `MaxIte` is the maximum number of iterations. After `MaxIte` main iterations, GenOpt terminates with an error message.
`WriteStepNumber` specifies whether the current step of the optimization has to be written to the simulation input file or to a function object. The step number can be used to calculate a penalty or barrier function (see Section 8.2 on page 65).
The optional parameter `MaxEqualResults` specifies how many times the cost function value can be equal to a value that has previously been obtained before GenOpt terminates. This setting is used to terminate GenOpt if the cost function value is constant for several iterates (see Section 11.4). The default value of `MaxEqualResults` is 5.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

The optional parameter `UnitsOfExecution` specifies the maximum number of simulations that may run in parallel. If this parameter is not specified or set to zero, then its value is set to the number of processors of the computer that runs GenOpt. In general, this parameter need not be specified.

`Algorithm` The setting of `Main` specifies which algorithm is invoked for doing the optimization. Its value has to be equal to the class name that contains the algorithm. Note that additional parameters might be required depending on the algorithm used (see Section 5 for the implemented algorithms).

11.1.4 Log File

GenOpt writes a log file to the directory that contains the initialization file. The name of the log file is `GenOpt.log`.

The GenOpt log file contains general information about the optimization process. It also contains warnings and errors that occur during the optimization.

11.1.5 Output Files

In addition to `GenOpt.log`, GenOpt writes two output files to the directory where the optimization command file is located. (The location of the optimization command file is defined by the variable `Optimization.Files.Command.Path1` in the optimization initialization file.)

The iterations are written to the output files `OutputListingMain.txt` and `OutputListingAll.txt`. The file `OutputListingMain.txt` contains only the main iteration steps, whereas `OutputListingAll.txt` contains all iteration steps.

Each time the method `genopt.algorithm.Optimizer.report()` is called from the optimization algorithm, the current trial is reported in either one of the files.

11.2 Resolving Directory Names for Parallel Computing

To allow doing simulations using parallel computing, GenOpt will create a temporary directory for each simulation. This avoids different simulations writing to the same output or log files simultaneously. The simulations will be done in subdirectories of the directory that contains the optimization initialization file.

To explain which directories are created by GenOpt, suppose that GenOpt's optimization initialization file is stored in the directory `/data/optMacOSX.ini`. (For Windows, simply replace `/data` with `C:\data` and replace all forward-

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

slashes with backslashes.) Suppose that GenOpt's initialization file states that the simulation input file is `/data/input/in.txt`, the simulation log file is `/data/log.txt`, and the simulation output file is `/data/output/out.txt`. Thus, in this example, the simulation will read its input from `input/in.txt`, it will write its log messages to `log.txt`, and it will write its output to `output/out.txt`, where the directories `input` and `output` are subdirectories of the directory in which the simulation was started. Then, for the first simulation, GenOpt will proceed as follows:

1. It will create the simulation input file `/data/tmp-genopt-run-1/input/in.txt` (including the temporary directory `tmp-genopt-run-1/input`).
2. It will change the working directory for the simulation to the directory `/data/tmp-genopt-run-1`. Hence, if the simulation program writes to the current directory, then it will write to `/data/tmp-genopt-run-1`.
3. GenOpt will read `/data/tmp-genopt-run-1/log.txt` to retrieve the simulation log messages.
4. If no error has been found in the log file, then GenOpt will read the simulation output file `/data/tmp-genopt-run-1/output/out.txt`.
5. GenOpt will delete the directory `/data/tmp-genopt-run-1` and all its subdirectories.

For the second simulation, the same steps will be repeated, but the temporary directory will be `/data/tmp-genopt-run-2`.

To resolve the directory names, GenOpt uses the following rules. The rules are applied to the directories of the simulation input files, simulation log files and simulation output files. They are also applied to the value of the keyword `Command` in the section `SimulationStart` of the optimization configuration file.

1. A period (".") is replaced by the path name of the optimization initialization file.
2. If the keywords `Path1`, `Path2` etc. are not specified in the optimization initialization file, then they will be set to the directory of the optimization initialization file.
3. For the simulation input, the simulation log and the simulation output files, the string `tmp-genopt-run-#`, where `#` is the number of the simulation, will be inserted between the name of the optimization initialization file and the subdirectory name of the simulation input, log or output file.
4. When resolving the file names, a path separator ("\" on Windows or "/" on Mac OS X and Linux) will be appended if needed.

These rules work for situations in which the simulation program uses the current directory, or subdirectories of the current directory, to read input and write output, provided that the optimization configuration file is also in the directory that contains the simulation input files.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

For the declaration of the `Command` line in the GenOpt configuration file, we recommend using the full directory name. For example, we recommend using

```
Command = "./simulate.sh    [linebreak added]
  %Simulation.Files.Log.Path1%/%Simulation.Files.Log.File1%";
```

instead of

```
Command = "./simulate.sh ./%Simulation.Files.Log.File1%";
```

The first version ensures that the argument that is passed to `simulate.sh` is the simulation log file in the working directory that is used by the current simulation. However, in the second version, because of rule (1) the simulation log file will be in the directory of GenOpt's configuration file, and thus different simulations may write to the same simulation log file simultaneously, causing unpredictable behavior.

11.3 Pre-Processing and Post-Processing

Some simulation programs do not have the capability to pre-process the independent variables, or to post-process values computed during the simulation. For such situations, GenOpt's *input function objects* and *output function objects* can be used.

a) Function Objects

Function objects are formulas whose arguments can be the independent variables, the keyword `stepNumber`, and for post-processing, the result of the simulation.

Following functions are implemented:

Function	Returns
`add(x0, x1)`	$x^0 + x^1$
`add(x0, x1, x2)`	$x^0 + x^1 + x^2$
`subtract(x0, x1)`	$x^0 - x^1$
`multiply(x0, x1)`	$x^0\, x^1$
`multiply(x0, x1, x2)`	$x^0\, x^1\, x^2$
`divide(x0, x1)`	x^0/x^1
`log10(x0)`	$\log_{10}(x^0)$

Furthermore, all functions that are defined in the class `java.lang.StrictMath` and whose arguments and return type are of type `double` can be accessed by typing their name (without the package and class name).

In addition, users can implement any other static method with arguments and return type `double` by adding the method to `genopt/algorithm/util/math/Fun.java`. The method must have the syntax

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

```
public static double  methodName(double x0, double x1) {
  double r;
  //  do any computations
  return r;
}
```

The number of arguments is arbitrary.

Compile the file after adding new methods. No other changes are required. To compile the file, a Java compiler must be installed (such as the one from Sun Microsystems). To compile it, open a console (or DOS window), change to the directory `genopt/algorithm/util/math` and type

```
javac Fun.java
```

This will generate the file `Fun.class`. If the compilation fails, then the variable `CLASSPATH` is probably not set as described in Chapter 10.

Next, we present an example for pre-processing and afterwards an example for post-processing.

b) Pre-Processing

Example 11.3.1 Suppose we want to find the optimal window width and height. Let `w` and `h` denote the window width and height, respectively. Suppose we want the window height to be 1/2 times the window width, and the window width must be between 1 and 2 meters. Then, we could specify in the command file the section

```
Parameter{
   Name =   w;
   Ini  = 1.5;     Step = 0.05;
   Min  =   1;     Max  =    2;
   Type = CONTINUOUS;
}
Function{
   Name =   h;     Function = "multiply( %w%, 0.5 )";
}
```

Then, in the simulation input template files, GenOpt will replace all occurrences of `%w%` by the window width and all occurences of `%h%` by 1/2 times the numerical value of `%w%`. □

GenOpt does not report values that are computed by input functions. To report such values, a user needs to specify them in the section `ObjectiveFunctionLocation`, as shown in Example 11.3.2 below.

c) Post-Processing

Example 11.3.2 Suppose we want to minimize the sum of annual heating and cooling energy consumption, which we will call *total energy*. Some simulation programs cannot add different output variables. For example, EnergyPlus [CLW+01] writes the heating and cooling energy consumption separately

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

to the output file. In order to optimize the total energy, the simulation output must be post-processed.

To post-process the simulation results in GenOpt, we can proceed as follows:
Suppose the cost function delimiter (see Section 11.1.1) for the heating and cooling energy are, respectively, `Eheat=` and `Ecool=`. In addition, suppose we want to report the value of the variable `h` that has been computed by the input function object in Example 11.3.1.

Then, in the optimization initialization file (see Section 11.1.1) we can specify the section

```
ObjectiveFunctionLocation{
   Name1 = E_tot;  Function1 = "add( %E_heat%, %E_cool% )";
   Name2 = E_heat; Delimiter2 = "Eheat=";
   Name3 = E_cool; Delimiter3 = "Ecool=";
   Name4 = height; Function4  = %h%;
}
```

This specification causes GenOpt to (i) substitute the value of `h` in `Function4`, (ii) read from the simulation output file(s) the numbers that occur after the strings `Eheat=` and `Ecool=`, (iii) substitute these numbers into the function `add( %E_heat%, %E_cool% )`, (iv) evaluate the functions `Function1` and `Function4`, and (v) minimize the sum of heating and cooling energy. □

As arguments of the function defined above, we can use any name of an independent variable, of an input function object, or the keyword `%stepNumber%`.

11.4 Truncation of Digits of the Cost Function Value

For $x' \in \mathbb{R}^n$ and $f\colon \mathbb{R}^n \to \mathbb{R}$, assume there exists a scalar $\delta > 0$ such that $f(x') = f(x'')$ for all $x'' \in B(x', \delta)$, where $B(x', \delta) \triangleq \{x'' \in \mathbb{R}^n \mid \|x' - x''\| < \delta\}$. Obviously, in $B(x', \delta)$, an optimization algorithm can fail because iterates in $B(x', \delta)$ contain no information about descent directions outside of $B(x', \delta)$. Furthermore, in absence of convexity of $f(\cdot)$, the optimality of x' cannot be ascertain in general.

Such situations can be generated if the simulation program writes the cost function value to the output file with only a few digits. Fig. 11.1 illustrates that truncating digits can cause problems particularly in domains of $f(\cdot)$ where the slope of $f(\cdot)$ is flat. In Fig. 11.1, we show the function

$$f(x) \triangleq 0.1\,x - 0.1\,x^2 + 0.04\,x^4. \tag{11.1}$$

The upper line is the exact value of $f(\cdot)$, and the lower line is the rounded value of $f(\cdot)$ such that it has only two digits beyond the decimal point. If the

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

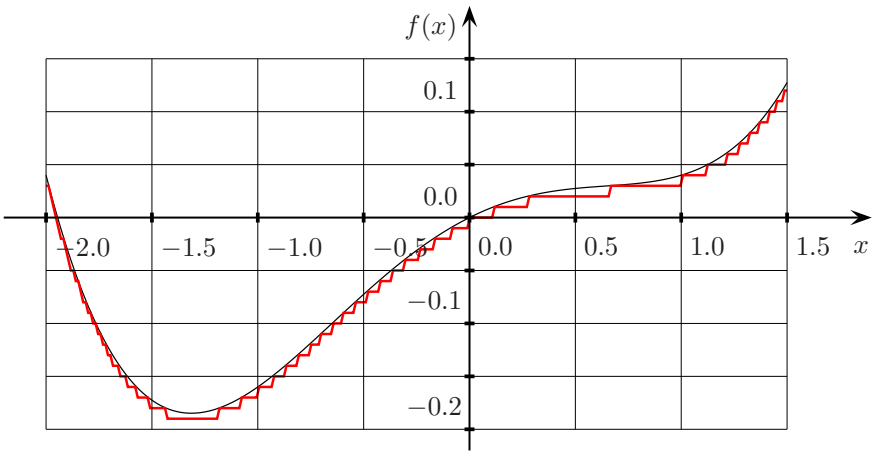


Figure 11.1: Function (11.1) *with machine precision and with truncated digits. The upper line shows the cost function value with machine precision, and the lower line shows the cost function value with only two digits beyond the decimal point.*

optimization algorithm makes changes in x in the size of 0.2, then it may fail for $0.25 < x < 1$, which is far from the minimum. In this interval, no useful information about the descent of $f(\cdot)$ can be obtained. Thus, the cost function must be written to the output file without truncating digits.

To detect such cases, the optimization algorithm can cause GenOpt to check whether a cost function values is equal to a previous cost function value. If the same cost function value is obtained more than a user-specified number of times, then GenOpt terminates with an error message. The maximum number of equal cost function values is specified by the parameter `MaxEqualResults` in the command file (see page 83).

GenOpt writes an information to the user interface and to the log file if a cost function value is equal to a previous function value.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

12 Conclusion

In system optimization it is not possible to apply a general optimization algorithm that works efficiently on all problems. What algorithm should be used depends on the properties of the cost function, such as the number of independent parameters, the continuity of the cost function and its derivatives, and the existence of local minima. Thus a variety of optimization algorithms is needed. To address this need, GenOpt has a library with different optimization algorithms and an optimization algorithm interface that users can use to implement their own optimization algorithm if desired.

The fact that analytical properties of the cost function are unavailable for the class of optimization problems that GenOpt has been developed for makes it possible to separate optimization and function evaluation. Therefore, GenOpt has a simulation program interface that allows coupling any program that exchanges input and output using text files. Hence, users are not restricted to using a special program for evaluating the cost function. Rather, they can use the simulation program they are already using for their system design and development. Thus, the system can be optimized with little additional effort.

This open environment not only allows coupling any simulation program and implementing special purpose algorithms, but it also allows sharing algorithms among users. This makes it possible to extend the algorithm library and thus extend GenOpt's applicability.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

13 Acknowledgment

The development of GenOpt was sponsored by grants from the Swiss Academy of Engineering Sciences (SATW), the Swiss National Energy Fund (NEFF) and the Swiss National Science Foundation (SNF) and is supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technology Programs of the U.S. Department of Energy, under Contract No. DE-AC03-76SF00098. I would like to thank these institutions for their generous support.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

14 Legal

14.1 Copyright Notice

GenOpt Copyright (c) 1998-2008, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from the U.S. Dept. of Energy). All rights reserved.

If you have questions about your rights to use or distribute this software, please contact Berkeley Lab's Technology Transfer Department at `TTD@lbl.gov`.

NOTICE. This software was developed under partial funding from the U.S. Department of Energy. As such, the U.S. Government has been granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable, worldwide license in the Software to reproduce, prepare derivative works, and perform publicly and display publicly. Beginning five (5) years after the date permission to assert copyright is obtained from the U.S. Department of Energy, and subject to any subsequent five (5) year renewals, the U.S. Government is granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable, worldwide license in the Software to reproduce, prepare derivative works, distribute copies to the public, perform publicly and display publicly, and to permit others to do so.

14.2 License agreement

GenOpt Copyright (c) 1998-2008, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from the U.S. Dept. of Energy). All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University of California, Lawrence Berkeley National Laboratory, U.S. Dept. of Energy nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

You are under no obligation whatsoever to provide any bug fixes, patches, or upgrades to the features, functionality or performance of the source code ("Enhancements") to anyone; however, if you choose to make your Enhancements available either publicly, or directly to Lawrence Berkeley National Laboratory, without imposing a separate written license agreement for such Enhancements, then you hereby grant the following license: a non-exclusive, royalty-free perpetual license to install, use, modify, prepare derivative works, incorporate into other computer software, distribute, and sublicense such enhancements or derivative works thereof, in binary and source code form.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

A Benchmark Tests

This section lists the settings used in the benchmark tests on page 51.

The settings in `OptimizationsSettings` and `Algorithm` are the same for all runs expect for `Accuracy`, which is listed in the result chart on page 52.

The common settings were:

```
OptimizationSettings{
    MaxIte          = 1500;
    WriteStepNumber = false;
}

Algorithm{
   Main = NelderMeadONeill;
   Accuracy =  see page 52;
   StepSizeFactor = 0.001;
   BlockRestartCheck = 5;
   ModifyStoppingCriterion  =  see page 52;
}
```

The benchmark functions and the `Parameter` settings in the `Vary` section are shown below.

A.1 Rosenbrock

The Rosenbrock function that is shown in Fig A.1 is defined as

$$f(x) \triangleq 100\left(x^2 - (x^1)^2\right)^2 + (1 - x^1)^2, \tag{A.1}$$

where $x \in \mathbb{R}^2$. The minimum is at $x^* = (1, 1)$, with $f(x^*) = 0$.

The section `Vary` of the optimization command file was set to

```
Vary{
   Parameter{
      Name =   x1; Min = SMALL;
      Ini  = -1.2; Max = BIG;
      Step =    1;
      }
   Parameter{
      Name =   x2; Min = SMALL;
      Ini  =    1; Max = BIG;
      Step =    1;
      }
}
```

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

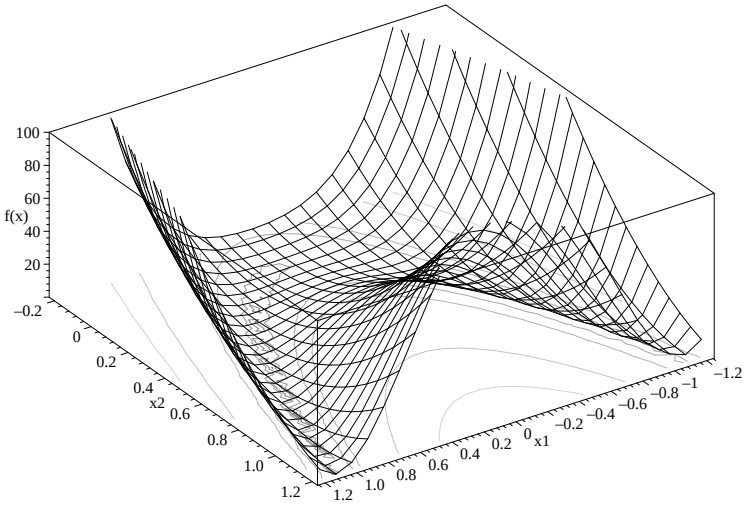


Figure A.1: *Rosenbrock function.*

A.2 Function 2D1

This function has only one minimum point. The function is defined as

$$f(x) \triangleq \sum_{i=1}^{3} f^i(x), \tag{A.2}$$

with

$$f^1(x) \triangleq \langle b,\, x\rangle + \frac{1}{2}\langle x,\, Q\, x\rangle, \quad b \triangleq \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad Q \triangleq \begin{pmatrix} 10 & 6 \\ 6 & 8 \end{pmatrix}, \tag{A.3}$$

$$f^2(x) \triangleq 100 \arctan\big((2 - x^1)^2 + (2 - x^2)^2\big), \tag{A.4}$$

$$f^3(x) \triangleq 50 \arctan\big((0.5 + x^1)^2 + (0.5 + x^2)^2\big), \tag{A.5}$$

where $x \in \mathbb{R}^2$. The function has a minimum at $x^* = (1.855340, 1.868832)$, with $f(x^*) = -12.681271$. It has two regions where the gradient is very small (see Fig. A.2).

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

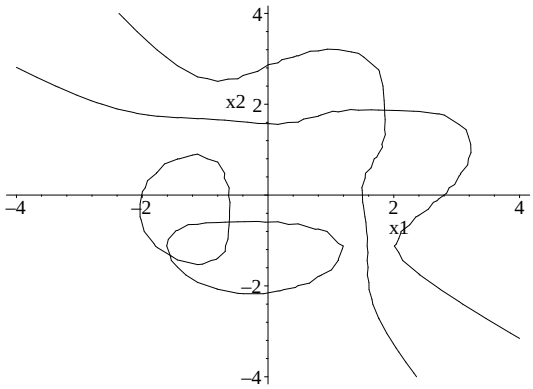


Figure A.2: Contour plot of $\frac{\partial df(x)}{\partial x^1} = 0$ *and* $\frac{\partial df(x)}{\partial x^2} = 0$*, where* $f(x)$ *is as in (A.2).*

The section `Vary` of the optimization command file is

```
Vary{
   Parameter{
      Name = x0; Min = SMALL;
      Ini  = -3; Max = BIG;
      Step = 0.1;
   }
   Parameter{
      Name =  x1; Min = SMALL;
      Ini  =  -3; Max = BIG;
      Step = 0.1;
   }
}
```

A.3 Function Quad

The function "Quad" is defined as

$$f(x) \triangleq \langle b,\, x\rangle + \frac{1}{2}\langle x,\, M\, x\rangle, \tag{A.6}$$

where $b, x \in \mathbb{R}^{10}$, $M \in \mathbb{R}^{10\times 10}$, and

$$b \triangleq (10,\, 10,\, \ldots\, ,\, 10)^T. \tag{A.7}$$

This function is used in the benchmark test with two different positive definite matrices M. In one test case, M is the identity matrix I and in the other test case M is a matrix, called Q, with a large range of eigenvalues. The matrix Q has elements

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

579.7818	−227.6855	49.2126	−60.3045	−152.4101	−207.2424	8.0917	33.6562	204.1312	−3.7129
−227.6855	236.2505	−16.7689	−40.3592	179.8471	80.0880	−64.8326	15.2262	−92.2572	40.7367
49.2126	−16.7689	84.1037	−71.0547	20.4327	5.1911	−58.7067	−36.1088	−62.7296	7.3676
−60.3045	−40.3592	−71.0547	170.3128	−140.0148	8.9436	26.7365	125.8567	62.3607	−21.9523
−152.4101	179.8471	20.4327	−140.0148	301.2494	45.5550	−31.3547	−95.8025	−164.7464	40.1319
−207.2424	80.0880	5.1911	8.9436	45.5550	178.5194	22.9953	−39.6349	−88.1826	−29.1089
8.0917	−64.8326	−58.7067	26.7365	−31.3547	22.9953	124.4208	−43.5141	75.5865	−32.2344
33.6562	15.2262	−36.1088	125.8567	−95.8025	−39.6349	−43.5141	261.7592	86.8136	22.9873
204.1312	−92.2572	−62.7296	62.3607	−164.7464	−88.1826	75.5865	86.8136	265.3525	−1.6500
−3.7129	40.7367	7.3676	−21.9523	40.1319	−29.1089	−32.2344	22.9873	−1.6500	49.2499

The eigenvalues of Q are in the range of 1 to 1000.

The functions have minimum points x^* at

Matrix M:	I	Q
${x^*}^0$	−10	−2235.1810
${x^*}^1$	−10	−1102.4510
${x^*}^2$	−10	790.6100
${x^*}^3$	−10	−605.2480
${x^*}^4$	−10	−28.8760
${x^*}^5$	−10	228.7640
${x^*}^6$	−10	−271.8830
${x^*}^7$	−10	−3312.3890
${x^*}^8$	−10	−2846.7870
${x^*}^9$	−10	−718.1490
$f(x^*)$	−500	0

Both test functions have been optimized with the same parameter settings. The settings for the parameters `x0` to `x9` are all the same, and given by

```
Vary{
   Parameter{
      Name = x0; Min = SMALL;
      Ini  =  0; Max = BIG;
      Step =  1;
   }
}
```

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

Bibliography

[AD03] Charles Audet and John E. Dennis, Jr. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.

[BP66] M. Bell and M. C. Pike. Remark on algorithm 178. *Comm. ACM*, 9:685–686, September 1966.

[CD01] A. Carlisle and G. Dozier. An off-the-shelf PSO. In *Proceedings of the Workshop on Particle Swarm Optimization*, Indianapolis, IN, 2001.

[CK02] Maurice Clerc and James Kennedy. The particle swarm – explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, February 2002.

[CLW+01] Drury B. Crawley, Linda K. Lawrie, Frederick C. Winkelmann, Walter F. Buhl, Y. Joe Huang, Curtis O. Pedersen, Richard K. Strand, Richard J. Liesen, Daniel E. Fisher, Michael J. Witte, and Jason Glazer. EnergyPlus: creating a new-generation building energy simulation program. *Energy and Buildings*, 33(4):443–457, 2001.

[DV68] R. De Vogelaere. Remark on algorithm 178. *Comm. ACM*, 11:498, July 1968.

[EK95] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Nagoya, Japan, October 1995. IEEE.

[ES01] R. C. Eberhart and Y. Shi. Particle swarm optimization: Developments, applications and resources. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 1, pages 84–86, Seoul, South Korea, 2001. IEEE.

[HJ61] R. Hooke and T. A. Jeeves. 'Direct search' solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8(2):212–229, 1961.

[KDB76] S. A. Klein, J. A. Duffie, and W. A. Beckman. TRNSYS – A transient simulation program. *ASHRAE Transactions*, 82(1):623–633, 1976.

[KE95] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Perth, Australia, November 1995.

[KE97] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proc. of Systems, Man, and Cybernetics*, volume 5, pages 4104–4108. IEEE, October 1997.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

[Kel99a] C. T. Kelley. Detection and remediation of stagnation in the Nelder–Mead algorithm using a sufficient decrease condition. *SIAM Journal on Optimization*, 10(1):43–55, 1999.

[Kel99b] C. T. Kelley. *Iterative methods for optimization*. Frontiers in Applied Mathematics. SIAM, 1999.

[KES01] James Kennedy, Russell C. Eberhart, and Yuhui Shi. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[KLT03] Tamara G. Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.

[KM02] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1671–1676. IEEE, 2002.

[LPV02a] E. C. Laskari, K. E. Parsopoulos, and M. N. Vrahatis. Particle swarm optimization for integer programming. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1582–1587, Honolulu, HI, May 2002. IEEE.

[LPV02b] E. C. Laskari, K. E. Parsopoulos, and M. N. Vrahatis. Particle swarm optimization for minimax problems. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1576–1581, Honolulu, HI, May 2002. IEEE.

[LRWW98] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147, 1998.

[McK98] K. I. M. McKinnon. Convergence of the Nelder-Mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158, 1998.

[NM65] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, January 1965.

[O'N71] R. O'Neill. Algorithm AS 47 – Function minimization using a simplex procedure. *Appl. Stat. 20*, 20:338–345, 1971.

[PFTV93] W. H. Press, B. P. Flannery, S. A. Tuekolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*, chapter 20. Cambridge University Press, 1993.

[Pol97] Elijah Polak. *Optimization, Algorithms and Consistent Approximations*, volume 124 of *Applied Mathematical Sciences*. Springer Verlag, 1997.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

[PV02a] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method for constrained optimization problems. In P. Sincak, J. Vascak, V. Kvasnicka, and J. Pospichal, editors, *Intelligent Technologies – Theory and Applications: New Trends in Intelligent Technologies*, volume 76 of *Frontiers in Artificial Intelligence and Applications*, pages 214–220. IOS Press, 2002. ISBN: 1-58603-256-9.

[PV02b] K. E. Parsopoulos and M. N. Vrahatis. Recent approaches to global optimization problems through Particle Swarm Optimization. *Natural Computing*, 1:235–306, 2002.

[PW03] Elijah Polak and Michael Wetter. Generalized pattern search algorithms with adaptive precision function evaluations. Technical Report LBNL-52629, Lawrence Berkeley National Laboratory, Berkeley, CA, 2003.

[SE98] Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation*, IEEE World Congress on Computational Intelligence, pages 69–73, Anchorage, AK, May 1998. IEEE.

[SE99] Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 1945–1950, Carmel, IN, 1999. IEEE.

[Smi69] Lyle B. Smith. Remark on algorithm 178. *Comm. ACM*, 12:638, November 1969.

[Tor89] Virginia Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Rice University, Houston, TX, May 1989.

[Tse99] Paul Tseng. Fortified-descent simplicial search method: A general approach. *SIAM Journal on Optimization*, 10(1):269–288, 1999.

[vdBE01] F. van den Bergh and A.P Engelbrecht. Effects of swarm size on cooperative particle swarm optimisers. In *GECCO*, San Francisco, CA, July 2001.

[WBB+93] F. C. Winkelmann, B. E. Birsdall, W. F. Buhl, K. L. Ellington, A. E. Erdem, J. J. Hirsch, and S. Gates. DOE-2 supplement, version 2.1E. Technical Report LBL-34947, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, November 1993.

[WP03] Michael Wetter and Elijah Polak. A convergent optimization method using pattern search algorithms with adaptive precision simulation. In G. Augenbroe and J. Hensen, editors, *Proc. of the 8-th IBPSA Conference*, volume III, pages 1393–1400, Eindhoven, NL, August 2003.

[Wri96] M. H. Wright. Direct search methods: once scorned, now respectable. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1995*, pages 191–208. Addison Wesley Longman (Harlow), 1996.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.

[WW03] Michael Wetter and Jonathan Wright. Comparison of a generalized pattern search and a genetic algorithm optimization method. In G. Augenbroe and J. Hensen, editors, *Proc. of the 8-th IBPSA Conference*, volume III, pages 1401–1408, Eindhoven, NL, August 2003.

Copyright (c) 1998-2008
The Regents of the University of California (through Lawrence Berkeley National Laboratory),
subject to receipt of any required approvals from U.S. Department of Energy. All rights reserved.