Chapter 4

Equality-Constrained Optimization: The Orthogonal-Decomposition Algorithm

4.1 Introduction

The numerical solution of equality-constrained problems is the subject of this chapter. In this vein, we focus on methods stemming from nonlinear least-square problems, that lead to what is known as *sequential quadratic programmiong* (SQP). SQP appears to be the most commonly used and reliable method in nonlinear programming, for it is well suited for the solution of nonlinear programming problems, as reported by Murray (1997) and Lalee, Nocedal and Plantenga (1998). In SQP, the nonlinear optimization problem is approximated by a sequence of quadratic programs (QP), each being a sub-quadratic program (sub-QP).

Motivated by nonlinear least-square problems, QP works under the assumption that the feasible Hessian is positive-definite at each iteration. If this is not the case at a given iteration, then the Hessian matrix is modified to render it so, a procedure called *Hessian-stabilization*. Then, the sub-QP is solved by means of any algorithm suitable for QP problems. The procedure is terminated as a criterion is met with a prescribed tolerance. The solution procedure of SQP thus involves two phases: the stabilization of the Hessian matrix and the QP solution. The Hessian matrix can be stabilized by methods such as that proposed by Broyden (1970), Fletcher (1970), Goldfarb (1970) and Shanno (1970), which is known as the BFGS method. The BFGS method, implemented in the Matlab Optimization Toolbox, is thought to be very effective for use in general applications and thus, appears to be the most popular. In addition, methods for solving QP problems are for example, the coordinate-ascent method (Bertsekas, 1995) and quasi-Newton methods (Rao, 1996).

The main item introduced in this chapter is the *orthogonal decomposition algorithm* (ODA), which is derived first in the context of equality-constrained linear least-square problems; then, it is applied to equality-constrained nonlinear least-square problems. Several numerical techniques, such as Householder reflections, Cholesky decomposition, the Newton-Gauss method, etc., are applied in order to obtain numerical solutions by means of procedures that are both *efficient* and *robust*. What we mean by the former is procedures that use as few floating-point operations (flops) as possible; by the latter we mean procedures that keep the roundoff error in the solution as low as possible with respect to that of the data, an item that falls in the realm of *numerical conditioning*.

The orthogonal-decomposition algorithm is implemented in a C library of routines, called ODA, in combination with *Gerschgorin stabilization* (Teng and Angeles, 2001) for arbitrary objective functions, in the framework of sequential quadratic programming (SQP). Gerschgorin stabilization is based on the Gerschgorin Theorem (Varga, 2000), which provides a region of the complex plane in which the eigenvalues of an arbitrary $n \times n$ matrix are bound to lie. Moreover, the ODA is applied in solving the underlying sub-QP.

4.2 Equality-Constrained Linear Least-Square Problems: The Orthogonal-Decomposition Algorithm

We recall below the linear least-square problem subject to linear equality constraints: Given the overdetermined system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.1}$$

find a vector $\mathbf{x}$ that verifies the above system with the least-square error, which is defined as

$$f \equiv \frac{1}{2}(\mathbf{A}\mathbf{x} - \mathbf{b})^T \mathbf{W}(\mathbf{A}\mathbf{x} - \mathbf{b}) \to \min_{\mathbf{x}} \tag{4.2}$$

subject to the linear constraints

$$\mathbf{Cx} = \mathbf{d} \tag{4.3}$$

Here, $\mathbf{x}$ is the n-dimensional vector of design variables, while $\mathbf{A}$ and $\mathbf{C}$ are $q \times n$ and $p \times n$ matrices, while $\mathbf{b}$ and $\mathbf{d}$ are q- and p-dimensional vectors. Moreover, $\mathbf{W}$ is a $q \times q$ positive-definite weighting matrix, with q, p and n subject to

$$q > n \quad \text{and} \quad p < n \tag{4.4}$$

Note that the first of the foregoing inequalities excludes the possibility of a unique solution upon solving for $\mathbf{x}$ from eq.(4.1), the second preventing a unique solution from eq.(4.3).

If $\mathbf{A}$ and $\mathbf{C}$ are full-rank matrices, then the forgoing problem was shown to have a *unique* solution, eqs.(3.83a–d), reproduced below for quick reference:

$$\mathbf{x} = \mathbf{PQb} + \mathbf{Rd} \tag{4.5a}$$

where $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{R}$ are the $n \times n$-, $n \times q$- and $n \times p$ matrices that follow:

$$\mathbf{P} = \mathbf{1}_n - \mathbf{RC} \tag{4.5b}$$

$$\mathbf{Q} = (\mathbf{A}^T\mathbf{WA})^{-1}\mathbf{A}^T\mathbf{W} \tag{4.5c}$$

$$\mathbf{R} = (\mathbf{A}^T\mathbf{WA})^{-1}\mathbf{C}^T[\mathbf{C}(\mathbf{A}^T\mathbf{WA})^{-1}\mathbf{C}^T]^{-1} \tag{4.5d}$$

and $\mathbf{1}_n$ is the $n \times n$ identity matrix.

As pointed out in Subsection 3.4.2, the above expression is unsuitable for numerical implementation. A common alternative approach to obtain the solution under study consists in partitioning $\mathbf{C}$ into a $p \times p$ and a $p \times (n-p)$ submatrices, where care should be taken so as to choose a well-conditioned $p \times p$ matrix, for safe inversion. Correspondingly, vector $\mathbf{x}$ should be partitioned into a *master* part $\mathbf{x}_M$, of $n - p$ components, and a *slave* part $\mathbf{x}_S$ of p components. Thus, the constraint equations would be solved for the slave part in terms of the master part and the problem would reduce to an unconstrained least-square problem of dimension $n-p$. However, an arbitrary partitioning of $\mathbf{C}$ may lead to an ill-conditioned $p \times p$ block, even if $\mathbf{C}$ itself is well-conditioned. This situation can be prevented if, out of all N possible partitionings of $\mathbf{C}$, the one with the lowest condition number is chosen. Note that the number of partitionings is given by

$$N = \frac{n!}{p!(n-p)!}$$

and hence, N can become quite large, even for modest values of n and p. Since calculating the condition number of a matrix is a computationally costly procedure, this approach does not seem very attractive.

Alternatively, by introduction of the singular values of $\mathbf{C}$ (Strang, 1988), a subsystem of p equations in p unknowns, which are linear combinations of the components of $\mathbf{x}$, can be found that is optimally conditioned. The computation of singular values, however, similar to that of eigenvalues, is a problem even more difficult to solve than the one at hand, for it is nonlinear and must be solved iteratively. Therefore, it is not advisable to follow the singular-value approach either.

One more approach is followed here, which stems from the geometrical interpretation of the solution (4.5a). Indeed, vector $\mathbf{Qb}$ of that solution represents the unconstrained least-square approximation of eq.(4.2). The second term of the right-hand side of eq.(4.5a) is the minimum-norm solution of the underdetermined system (4.3), based on the norm defined as

$$\|\mathbf{x}\|_W^2 = \mathbf{x}^T \mathbf{A}^T \mathbf{W} \mathbf{A} \mathbf{x} \tag{4.6}$$

Thus, $\mathbf{P}$ is a *projector*[1] onto the nullspace of $\mathbf{C}$. Indeed, one can readily prove that every n-dimensional vector $\mathbf{x}$ is mapped by $\mathbf{P}$ onto the nullspace of $\mathbf{C}$. Moreover, $\mathbf{P}^2$ can be proven to equal $\mathbf{P}$, thereby making apparent that $\mathbf{P}$ is, in fact, a projector. Thus, the range and the nullspace of $\mathbf{C}$ appear to play an important role in the solution of the foregoing problem. Moreover, the range and the nullspace of any matrix representing a linear transformation of $\mathbb{R}^n$ are orthogonal subspaces of $\mathbb{R}^n$, their direct sum producing all of $\mathbb{R}^n$; i.e., every n-dimensional vector $\mathbf{x}$ can be *uniquely* decomposed into a vector lying in the range of $\mathbf{C}^T$ and a second one lying in the nullspace of $\mathbf{C}$. Now let $\mathbf{L}$ be an $n \times (n-p)$ matrix spanning the nullspace of $\mathbf{C}$, i.e.,

$$\mathbf{CL} = \mathbf{O}_{pn'} \tag{4.7}$$

where $\mathbf{O}_{pn'}$ represents the $p \times (n-p)$ zero matrix. Matrix $\mathbf{L}$ is known as an *orthogonal complement* of matrix $\mathbf{C}$. Thus, the solution to the above problem can be decomposed into two parts, namely,

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{x}_U \tag{4.8}$$

in which $\mathbf{x}_0$ represents the minimum-norm solution to the constraint equation (4.3), i.e., $\mathbf{x}_0$ lies in the range of $\mathbf{C}^T$, while $\mathbf{x}_U$ lies in the nullspace of $\mathbf{C}$. Vector $\mathbf{x}_0$ is com-

[1] Note that $\mathbf{P}$ is apparently not symmteric!

puted by means of an orthogonalization method rendering $\mathbf{C}^T$ in upper-triangular form, as discussed in Subsection 3.4.1, while vector $\mathbf{x}_U$ is computed by means of a linear least-square problem. We outline below the computation of $\mathbf{x}_U$.

Let us define a $q \times q$ matrix $\mathbf{V}$ as the Cholesky factor of the given weighting matrix $\mathbf{W}$, i.e.,

$$\mathbf{W} = \mathbf{V}^T\mathbf{V}$$

] Moreover, with $\mathbf{x}_0$ known, $\mathbf{x}_U$ is found as the least-square approximation of

$$\mathbf{VAx}_U = \mathbf{V}(\mathbf{b} - \mathbf{Ax}_0) \tag{4.9}$$

subject to the constraints

$$\mathbf{Cx}_U = \mathbf{0} \tag{4.10}$$

Further, let us represent $\mathbf{x}_U$ as the image of a $(n-p)$-dimensional vector under a transfromation given by an $(n-p) \times n$ matrix $\mathbf{L}$, namely,

$$\mathbf{x}_U = \mathbf{Lu} \tag{4.11}$$

with $\mathbf{L}$ defined, in turn, as introduced in eq.(4.7). Equation (4.9) thus becomes

$$\mathbf{VALu} = \mathbf{V}(\mathbf{b} - \mathbf{Ax}_0) \tag{4.12}$$

which is an overdetermined system of n linear equations in $n-p$ unknowns. It is thus apparent that $\mathbf{u}$ can be computed as the *unconstrained* least-square solution of eq.(4.12).

However, matrix $\mathbf{L}$, an orthogonal complement of $\mathbf{C}$, *is not unique*. We have thus reached a crucial point in the solution of the constrained linear least-square problem at hand: How to define $\mathbf{L}$. While $\mathbf{L}$ can be defined in infinitely-many forms—notice that, once *any* $\mathbf{L}$ has been found, a multiple of it also satisfies eq.(4.7). We define here a distinct $\mathbf{L}$ such that

$$\mathbf{HL} = \begin{bmatrix} \mathbf{O} \\ \mathbf{1} \end{bmatrix} \tag{4.13}$$

where $\mathbf{1}$ is the $(n-p) \times (n-p)$ identity matrix and $\mathbf{O}$ is the $p \times (n-p)$ zero matrix, while $\mathbf{H}$ is defined as the product of Householder reflections rendering $\mathbf{C}^T$ in upper-triangular from—see Subsection 3.4.1. From eq.(4.13), one can obtain matrix $\mathbf{L}$

without any additional computations, for

$$\mathbf{L} = \mathbf{H}^T \begin{bmatrix} \mathbf{O} \\ \mathbf{1} \end{bmatrix} \tag{4.14}$$

whence it is apparent that $\mathbf{L}$ is isotropic, i.e., its condition number is equal to unity. This means that the generalized inverse $\mathbf{L}^I$ can be computed without roundoff-error amplification. In fact, this inverse reduces to $\mathbf{L}^T$, for

$$\mathbf{L}^I = ([\mathbf{O}^T \quad \mathbf{1}]\mathbf{H}^T\mathbf{H}\begin{bmatrix} \mathbf{O} \\ \mathbf{1} \end{bmatrix})^{-1}[\mathbf{O}^T \quad \mathbf{1}]\mathbf{H}^T = \mathbf{L}^T \tag{4.15}$$

Once $\mathbf{L}$ is known, equation (4.12) can be solved for $\mathbf{u}$ as the least-square approximation of that system. Then, $\mathbf{x}_U$ is calculated from equation (4.11).

As the reader can readily prove, the two components of $\mathbf{x}$, $\mathbf{x}_U$ and $\mathbf{x}_L$, are orthogonal. For this reason, the foregoing procedure is known as the *Orthogonal-Decomposition Algorithm* (ODA).

4.3 Equality-Constrained Nonlinear Least-Square Problems

The solution of nonlinear least-square problems by means of the ODA is now straightforward: The problem consists in finding the least-square error f of an overdetermined system of nonlinear equations, $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{0}$, i.e.,

$$f(\mathbf{x}) = \frac{1}{2}\boldsymbol{\phi}^T\mathbf{W}\boldsymbol{\phi} \quad \rightarrow \quad \min_{\mathbf{x}} \tag{4.16a}$$

subject to the nonlinear constraints

$$\mathbf{h}(\mathbf{x}) = \mathbf{0} \tag{4.16b}$$

where $\boldsymbol{\phi}$ and $\mathbf{x}$ are q- and n-dimensional vectors, respectively, with $q > n$, and $\mathbf{W}$ is a $q \times q$ positive-definite weighting matrix. Moreover, $\mathbf{h}$ is a l-dimensional vector of nonlinear constraints.

The normality condition of the foregoing constrained problem was derived in Ch. 3 in its dual form, eq.(3.88), and recalled below for quick reference:

$$\mathbf{L}^T\boldsymbol{\Phi}^T\mathbf{W}\boldsymbol{\phi} = \mathbf{0}_{n'} \tag{4.17}$$

with $\mathbf{0}_{n'}$ denoting the $(n-l)$-dimensional zero vector.

The solution of the problem at hand is obtained iteratively: From an initial guess $\mathbf{x}^0$, *not necessarily feasible*, i.e., with $\mathbf{h}(\mathbf{x}^0) \neq \mathbf{0}$, the sequence $\mathbf{x}^1$, $\mathbf{x}^2$, ..., $\mathbf{x}^k$, $\mathbf{x}^{k+1}$ is generated as

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x}^k \tag{4.18}$$

The increment $\Delta\mathbf{x}^k$ is computed as the solution of an equality-constrained linear least-square problem, namely,

$$\min_{\Delta\mathbf{x}^k} \frac{1}{2}\mathbf{e}^{k^T}\mathbf{W}\mathbf{e}^k \tag{4.19a}$$

subject to

$$\mathbf{J}(\mathbf{x}^k)\Delta\mathbf{x}^k = -\mathbf{h}(\mathbf{x}^k) \tag{4.19b}$$

with $\mathbf{e}^k$ defined as

$$\mathbf{e}^k \equiv -\boldsymbol{\phi}(\mathbf{x}^k) - \boldsymbol{\Phi}(\mathbf{x}^k)\Delta\mathbf{x}^k \tag{4.19c}$$

Now, for compactness, we introduce a few definitions:

$$\mathbf{h}^k \equiv \mathbf{h}(\mathbf{x}^k), \quad \boldsymbol{\phi}^k \equiv \boldsymbol{\phi}(\mathbf{x}^k), \quad \boldsymbol{\Phi}_k \equiv \boldsymbol{\Phi}(\mathbf{x}^k), \quad \mathbf{J}_k \equiv \mathbf{J}(\mathbf{x}^k) \tag{4.20}$$

while $\mathbf{L}_k$ is defined as the isotropic orthogonal complement of $\mathbf{J}_k$ à la eq.(4.13). Moreover, $\boldsymbol{\Phi}_k$ and $\mathbf{J}_k$ will be assumed to be of full rank throughout, the solution $\Delta\mathbf{x}^k$ of problem (4.19a–c) thus being expressed as

$$\Delta\mathbf{x}^k = \Delta\mathbf{v}^k + \mathbf{L}_k\Delta\mathbf{u}^k \tag{4.21}$$

where $\Delta\mathbf{v}^k$ and $\Delta\mathbf{u}^k$ are the minimum-norm and the least-square solutions to an underdetermined and an overdetermined system, namely,

$$\mathbf{J}_k\Delta\mathbf{v}^k = -\mathbf{h}^k \tag{4.22a}$$

$$\mathbf{V}\boldsymbol{\Phi}_k\mathbf{L}_k\Delta\mathbf{u}^k = \mathbf{V}(-\boldsymbol{\phi}^k - \boldsymbol{\Phi}_k\Delta\mathbf{v}^k) \tag{4.22b}$$

The stopping criteria of the procedure are, then,

$$||\Delta\mathbf{x}^k|| \leq \epsilon_1 \quad \text{and} \quad ||\mathbf{h}(\mathbf{x}^k)|| \leq \epsilon_2 \tag{4.23}$$

for prescribed tolerances ϵ_1 and ϵ_2. These criteria are verified when both the normality condition (4.17) *and the constraint* (4.16b) are verified within the given tolerances. Moreover, from eqs.(4.22a) and (4.22b), $\Delta\mathbf{x}^k$ can be expressed as

$$\Delta\mathbf{x}^k = (\mathbf{M}\boldsymbol{\Phi}^T\mathbf{W}\boldsymbol{\Phi} - \mathbf{1}_n)\mathbf{J}^\dagger\mathbf{h} - \mathbf{M}\boldsymbol{\Phi}^T\mathbf{W}\boldsymbol{\phi} \tag{4.24}$$

where subscripts and superscripts have been dropped from the right-hand side for compactness, $\mathbf{1}_n$ represents the $n \times n$ identity matrix, $\mathbf{J}^\dagger$ is the right Moore-Penrose generalized inverse of $\mathbf{J}$, and $\mathbf{M}$ is the $n \times n$ matrix defined as

$$\mathbf{M} = \mathbf{L}(\mathbf{L}^T \mathbf{\Phi}^T \mathbf{W} \mathbf{\Phi} \mathbf{L})^{-1} \mathbf{L}^T \tag{4.25}$$

Upon convergence, the constraint equations (4.16b) are verified, and hence, the first term of $\Delta \mathbf{x}^k$, as given by eq.(4.24), vanishes. Moreover, the normality condition (4.17) holds, $\Delta \mathbf{x}^k$ taking on the form shown below, upon convergence:

$$\Delta \mathbf{x}^k = \mathbf{L}(\mathbf{L}^T \mathbf{\Phi}^T \mathbf{W} \mathbf{\Phi} \mathbf{L})^{-1} \mathbf{L}^T \mathbf{J}^T \boldsymbol{\lambda} \tag{4.26}$$

However, $\mathbf{L}$ is the $n \times (n - l)$ isotropic orthogonal complement of $\mathbf{J}$, and hence,

$$\mathbf{J}\mathbf{L} = \mathbf{O}_{ln'} \tag{4.27}$$

with $\mathbf{O}_{ln'}$ defined as the $l \times (n - l)$ zero matrix. Therefore, upon convergence, $\Delta \mathbf{x}^k \rightarrow \mathbf{0}$.

The sequence $\{\Delta \mathbf{x}^k\}$ produces a sequence $\{f_k\}$, the increment Δf between two consecutive values of the sequence being given by

$$\Delta f = (\nabla f)^T \Delta \mathbf{x} \tag{4.28}$$

where ∇f is the gradient of f, i.e., $\nabla f = \mathbf{\Phi}^T \mathbf{W} \boldsymbol{\phi}$, and hence,

$$\begin{aligned} \Delta f &= (\mathbf{\Phi}^T \mathbf{W} \boldsymbol{\phi})^T \Delta \mathbf{x} \\ &= -\boldsymbol{\phi}^T \mathbf{W} \mathbf{\Phi} \mathbf{M} \mathbf{\Phi}^T \mathbf{W} \boldsymbol{\phi} - \boldsymbol{\phi}^T \mathbf{W} \mathbf{\Phi} (\mathbf{1} - \mathbf{M} \mathbf{\Phi}^T \mathbf{W} \mathbf{\Phi}) \mathbf{J}^\dagger \mathbf{h} \end{aligned} \tag{4.29}$$

From eq.(4.29), if the current value of $\mathbf{x}$ is feasible, i.e., if $\mathbf{h} = \mathbf{0}$, then Δf is negative definite, and the procedure yields an improved value of f. On the other hand, one can readily verify that

$$\Delta(\mathbf{h}^T \mathbf{h}) \equiv \mathbf{h}_{k+1} - \mathbf{h}_k = -2\mathbf{h}^T \mathbf{h} \tag{4.30}$$

which is negative definite. Therefore, the procedure gives a sequence of $\mathbf{x}$ values that approaches the constraints.

Example 4.3.1 (A Quadratic Objective Function with a Quadratic Constraint)

We recall Example 3.6.1, which is reproduced below for quick reference:

$$f(\mathbf{x}) = \frac{1}{2}(9x_1^2 - 8x_1x_2 + 3x_2^2) \quad \rightarrow \quad \min_{x_1, x_2}$$

subject to

$$h(\mathbf{x}) = x_1^2 + x_2^2 - 1 = 0$$

While the objective function is quadratic in a linear function of the design-variable vector, the constraint is nonlinear, which disqualifies this problem from a direct solution, as found in Section 4.2 for linear least-squares subject to linear constraints. This problem, due to its simplicity, could be solved exactly in Chapter 3. Here, we solve this problem numerically, using the ODA. First, note that the objective function $f(\mathbf{x})$ *can be factored as*

$$f(\mathbf{x}) = \frac{1}{2}\boldsymbol{\phi}^T\mathbf{W}\boldsymbol{\phi}$$

with

$$\mathbf{W} = \begin{bmatrix} 9 & -4 \\ -4 & 3 \end{bmatrix} \quad \text{and} \quad \boldsymbol{\phi} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

i.e, $f(\mathbf{x})$ *is a special case of the* $f(\mathbf{x})$ *defined in eq.(4.2), with* $\mathbf{A} = \mathbf{1}$ *and* $\mathbf{b} = \mathbf{0}$. *We include below a Maple worksheet describing the step-by-step implementation of the ODA in solving the foregoing problem* iteratively.

```
> restart:with(linalg):
```

```
Warning, the protected names norm and trace have been redefined and
unprotected
```

```
> with(plots): with(plottools):
```

```
Warning, the name changecoords has been redefined

Warning, the name arrow has been redefined
```

Example 4.3.1: Linear-least square problem subject to a quadratic constraint

$f(x)$ = (1/2)(9x_1^2 - 8 x_1 x_2 + 3x_2^2) —> min_{x_1, x_2} subject to

$$h(x\_1,\, x\_2) = x\_1\hat{}2 + x\_2\hat{}2 \text{ -1} = 0$$

```
> obj:= proc(x) (1/2)*(9*x[1]^2 - 8*x[1]*x[2] + 3*x[2]^2)
> end;#procedure to compute the objective function
```

$$obj := \mathbf{proc}(x)\, 9/2 * {x_1}^2 - 4 * x_1 * x_2 + 3/2 * {x_2}^2\, \mathbf{end\ proc}$$

```
> constr:= proc(x) x[1]^2+x[2]^2 - 1 end; #procedure computing the
> constraint
```

$$constr := \mathbf{proc}(x)\, {x_1}^2 + {x_2}^2 - 1\, \mathbf{end\ proc}$$

```
> dhdx:= proc(x) matrix([[2*x[1], 2*x[2]]]) end;#procedure computing
> the gradient of the constraint
```

$$dhdx := \mathbf{proc}(x)\, \mathrm{matrix}([[2 * x_1,\, 2 * x_2]])\, \mathbf{end\ proc}$$

```
> alfa:= proc(J) evalf(signum(J[1,1])*sqrt(J[1,1]^2 + J[1,2]^2))
> end;#procedure computing "alpha" of Householder reflections in
> least-square solution at each iteration
```

$$alfa := \mathbf{proc}(J)\, \mathrm{evalf}(\mathrm{signum}(J_{1,\,1}) * \mathrm{sqrt}({J_{1,\,1}}^2 + {J_{1,\,2}}^2))\, \mathbf{end\ proc}$$

```
> W:=matrix([[9, -4], [-4, 3]]);#weighting matrix
```

$$\mathrm{W} := \begin{bmatrix} 9 & -4 \\ -4 & 3 \end{bmatrix}$$

```
> V:=transpose(cholesky(W));#Maple returns a lower-triangular matrix
> with procedure "cholesky"!
```

$$V := \begin{bmatrix} 3 & \frac{-4}{3} \\ 0 & \frac{1}{3}\sqrt{11} \end{bmatrix}$$

```
> V:= map(evalf, V);
```

$$V := \begin{bmatrix} 3. & -1.333333333 \\ 0. & 1.105541597 \end{bmatrix}$$

```
> ID:=Matrix(2,2,shape=identity); E:= matrix([[0], [1]]); Phi:= ID;
> B:=evalm(V&*Phi);#Defining various auxiliary matrices
```

$$ID := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$E := \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\Phi := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$B := \begin{bmatrix} 3. & -1.333333333 \\ 0. & 1.105541597 \end{bmatrix}$$

```
> x:=vector([2, 2]); x0:= evalm(x);#initial guess, x^0, stored as x0
> for plotting
```

$$x := [2,\ 2]$$

$$x0 := [2,\ 2]$$

```
> f:= evalf(obj(x));#f_0
```

$$f := 8.$$

```
> phi:= evalm(x);#phi^0
```

$$\phi := [2,\ 2]$$

```
> h:= constr(x);#h^0
```

$$h := 7$$

```
> J:= dhdx(x);#J_0
```

$$J := \begin{bmatrix} 4 & 4 \end{bmatrix}$$

```
> alpha:= alfa(J);#local variable
```

$$\alpha := 5.656854248$$

```
> t:=vector([J[1,1] + alpha, J[1,2]]);#u in HHR algorithm, a local
> variable
```

$$t := [9.656854248,\ 4]$$

```
> normt2:=evalf(dotprod(t,t)/2);#half of Euclidean norm-squared of
t, a
> local variable
```

$$normt2 := 54.62741700$$

```
> H:=evalm(ID - t&*transpose(t)/normt2);#evaluating Householder
> reflection
```

$$H := \begin{bmatrix} -.707106781 & -.7071067812 \\ -.7071067812 & .7071067811 \end{bmatrix}$$

```
> P:=evalm(H&*transpose(H));#checking whether H is a reflection
```

$$P := \begin{bmatrix} .9999999997 & 0. \\ 0. & .9999999999 \end{bmatrix}$$

```
> detH:=det(H);
```

$$detH := -.9999999998$$

H is indeed a reflection!

```
> HJT:=evalm(H&*transpose(J));
```

$$HJT := \begin{bmatrix} -5.656854249 \\ -.1\,10^{-8} \end{bmatrix}$$

```
> HJT[2,1]:=0; print(HJT);#setting last entry of HJ^T equal to zero
```

$$HJT_{2,1} := 0$$

$$\begin{bmatrix} -5.656854249 \\ 0 \end{bmatrix}$$

```
> w:= vector([-h/HJT[1,1], 0]);#w = Hv
```

$$w := [1.237436867,\, 0]$$

```
> v:=evalm(H&*w);#v^0
```

$$v := [-.8749999997,\, -.8750000000]$$

```
> L:= evalm(H&*E); BL:=evalm(B&*L);# L_0 & (BL)_0
```

$$L := \begin{bmatrix} -.7071067812 \\ .7071067811 \end{bmatrix}$$

$$BL := \begin{bmatrix} -3.064129385 \\ .7817359600 \end{bmatrix}$$

```
> p:=matadd(phi, Phi&*v);#auxiliary variable
```

$$p := [1.125000000,\, 1.125000000]$$

```
> r:= evalm(-V&*p);#RHS of overdetermined system to compute u in ODA
```

$$r := [-1.875000000,\, -1.243734297]$$

```
> u:= leastsqrs(BL, r); #u^0
```

$$u := [.4772970772]$$

```
> Deltax:= matadd(v, L&*u);#Deltax^0
```

$$Deltax := [-1.212500000,\, -.5375000001]$$

First iteration is complete. Update **x**:

```
> x:= evalm(x + Deltax); x1:= evalm(x);#x^1
```

$$x := [.787500000,\, 1.462500000]$$

$$x1 := [.787500000,\, 1.462500000]$$

```
> f:= evalf(obj(x)); # f_1
```

$$f := 1.392187500$$

```
> phi:= evalm(x); # phi^1
```

$$\phi := [.787500000, 1.462500000]$$

```
> h:= constr(x); # h^1
```

$$h := 1.759062500$$

```
> J:= dhdx(x); # J_1
```

$$J := \begin{bmatrix} 1.575000000 & 2.925000000 \end{bmatrix}$$

```
> alpha:= alfa(J);
```

$$\alpha := 3.322085189$$

```
> t:=vector([J[1,1] + alpha, J[1,2]]);#u in HHR algorithm
```

$$t := [4.897085189, 2.925000000]$$

```
> normt2:=evalf(dotprod(t,t)/2);
```

$$normt2 := 16.26853418$$

```
> H:=evalm(ID - t&*transpose(t)/normt2);#evaluating Householder
> reflection
```

$$H := \begin{bmatrix} -.474099823 & -.8804710997 \\ -.8804710997 & .4740998233 \end{bmatrix}$$

```
> HJT:=evalm(H&*transpose(J));
```

$$HJT := \begin{bmatrix} -3.322085188 \\ .1\,10^{-8} \end{bmatrix}$$

```
> HJT[2,1]:=0; print(HJT);
```

$$HJT_{2,1} := 0$$

$$\begin{bmatrix} -3.322085188 \\ 0 \end{bmatrix}$$

```
> w:= vector([-h/HJT[1,1], 0]);#w = Hv
```

$$w := [.5295055366, 0]$$

```
> v:=evalm(H&*w); # v^1
```

$$v := [-.2510384812, -.4662143221]$$

```
> L:= evalm(H&*E); BL:=evalm(V&*Phi&*L); # L_1 & (BL)_1
```

$$L := \begin{bmatrix} -.8804710997 \\ .4740998233 \end{bmatrix}$$

$$BL := \begin{bmatrix} -3.273546397 \\ .5241370758 \end{bmatrix}$$

```
> p:=matadd(phi, Phi&*v);#auxiliary variable
```

$$p := [.5364615188, .9962856779]$$

```
> r:= evalm(-V&*p);#RHS of overdetermined system to compute u in ODA
```

$$r := [-.281003652, -1.101435259]$$

```
> u:= leastsqrs(BL, r); # u^1
```

$$u := [.03116921755]$$

```
> Deltax:= matadd(v, L&*u); # deltax^1
```

$$Deltax := [-.2784820764, -.4514370016]$$

Second iteration is complete. Update $\mathbf{x}$:

```
> x:= matadd(x, Deltax); x2:= evalm(x); # x^2
```

$$x := [.5090179236, 1.011062998]$$

$$x2 := [.5090179236, 1.011062998]$$

```
> f:= obj(x); # f_2
```

$$f := .640722436$$

```
> phi:= evalm(x); # phi^2
```

$$\phi := [.5090179236, 1.011062998]$$

```
> h:= constr(x); # h^2
```

$$h := .281347632$$

```
> J:= dhdx(x); # J_2
```

$$J := \begin{bmatrix} 1.018035847 & 2.022125996 \end{bmatrix}$$

```
> alpha:= alfa(J);
```

$$\alpha := 2.263932537$$

```
> t:=vector([J[1,1] + alpha, J[1,2]]);#u in HHR algorithm
```

$$t := [3.281968384, 2.022125996]$$

```
> normt2:=evalf(dotprod(t,t)/2);
```

$$normt2 := 7.430155005$$

```
> H:=evalm(ID - t&*transpose(t)/normt2);#evaluating Householder
> reflection
```

$$H := \begin{bmatrix} -.449675877 & -.8931918088 \\ -.8931918088 & .4496758759 \end{bmatrix}$$

```
> HJT:=evalm(H&*transpose(J));
```

$$HJT := \begin{bmatrix} -2.263932538 \\ -.12\,10^{-8} \end{bmatrix}$$

```
> HJT[2,1]:=0; print(HJT);
```

$$HJT_{2,1} := 0$$

$$\begin{bmatrix} -2.263932538 \\ 0 \end{bmatrix}$$

```
> w:= vector([-h/HJT[1,1], 0]);#w = Hv
```

$$w := [.1242738586, 0]$$

```
> v:=evalm(H&*w); # v^2
```

$$v := [-.05588295635, -.1110003925]$$

```
> L:= evalm(H&*E); BL:=evalm(V&*Phi&*L); # L_2 & (BL)_2
```

$$L := \begin{bmatrix} -.8931918088 \\ .4496758759 \end{bmatrix}$$

$$BL := \begin{bmatrix} -3.279143260 \\ .4971353860 \end{bmatrix}$$

```
> p:=matadd(phi, Phi&*v); # auxiliary variable
```

$$p := [.4531349672, .9000626055]$$

```
> r:= evalm(-V&*p); # RHS of overdetermined system to compute u in
> ODA
```

$$r := [-.159321428, -.9950566503]$$

```
> u:= leastsqrs(BL, r); # u^2
```

$$u := [.002523646038]$$

```
> Deltax:= matadd(v, L&*u); # Deltax^2
```

$$Deltax := [-.05813705632, -.1098655698]$$

Third iteration is complete. Update **x**:

```
> x:= matadd(x, Deltax); x3:=evalm(x);#x^3
```

$$x := [.4508808673, .9011974282]$$

$$x3 := [.4508808673, .9011974282]$$

```
> f:= obj(x); # f_3
```

$$f := .5077254992$$

```
> phi:= evalm(x); # phi^3
```

$$\phi := [.4508808673, .9011974282]$$

```
> h:= constr(x); # h^3
```

$$h := .015450361$$

```
> J:= dhdx(x); # J_3
```

$$J := \begin{bmatrix} .9017617346 & 1.802394856 \end{bmatrix}$$

```
> alpha:= alfa(J);
```

$$\alpha := 2.015391139$$

```
> t:=vector([J[1,1] + alpha, J[1,2]]);#u in HHR algorithm
```

$$t := [2.917152874, 1.802394856]$$

```
> normt2:=evalf(dotprod(t,t)/2);
```

$$normt2 := 5.879204055$$

```
> H:=evalm(ID - t&*transpose(t)/normt2);#evaluating Householder
> reflection
```

$$H := \begin{bmatrix} -.447437580 & -.8943151635 \\ -.8943151635 & .4474375804 \end{bmatrix}$$

```
> HJT:=evalm(H&*transpose(J));
```

$$HJT := \begin{bmatrix} -2.015391138 \\ .2\,10^{-9} \end{bmatrix}$$

```
> HJT[2,1]:=0; print(HJT);
```

$$HJT_{2,1} := 0$$

$$\begin{bmatrix} -2.015391138 \\ 0 \end{bmatrix}$$

```
> w:= vector([-h/HJT[1,1], 0]);#w = Hv
```

$$w := [.007666184846, 0]$$

```
> v:=evalm(H&*w); # v^3
```

$$v := [-.003430139195, -.006855985354]$$

```
> L:= evalm(H&*E); BL:=evalm(V&*Phi&*L); # L_3 & (BL)_3
```

$$L := \begin{bmatrix} -.8943151635 \\ .4474375804 \end{bmatrix}$$

$$BL := \begin{bmatrix} -3.279528930 \\ .4946608572 \end{bmatrix}$$

```
> p:=matadd(phi, Phi&*v);#auxiliary variable
```

$$p := [.4474507281, .8943414428]$$

```
> r:= evalm(-V&*p);#RHS of overdetermined system to compute u in ODA
```

$$r := [-.149896927, -.9887316669]$$

```
> u:= leastsqrs(BL, r); # u^3
```

$$u := [.0002276777133]$$

```
> Deltax:= matadd(v, L&*u); # Deltax^3
```

$$Deltax := [-.003633754826, -.006754113789]$$

Fourth iteration is complete. Update $\mathbf{x}$:

```
> x:= matadd(x, Deltax); x4:=evalm(x);# x^4
```

$$x := [.4472471125, .8944433144]$$

$$x4 := [.4472471125, .8944433144]$$

```
> f:= obj(x); # f_4
```

$$f := .5000294132$$

```
> phi:= evalm(x); # phi^4
```

$$\phi := [.4472471125, .8944433144]$$

```
> h:= constr(x); # h^4
```

$$h := .000058822$$

```
> J:= dhdx(x); # J_4
```

$$J := \begin{bmatrix} .8944942250 & 1.788886629 \end{bmatrix}$$

```
> alpha:= alfa(J);
```

$$\alpha := 2.000058822$$

```
> t:=vector([J[1,1] + alpha, J[1,2]]);#u in HHR algorithm
```

$$t := [2.894553047, 1.788886629]$$

```
> normt2:=evalf(dotprod(t,t)/2);
```

$$normt2 := 5.789276355$$

```
> H:=evalm(ID - t&*transpose(t)/normt2);#evaluating Householder
> reflection
```

$$H := \begin{bmatrix} -.447233960 & -.8944170093 \\ -.8944170093 & .4472339590 \end{bmatrix}$$

```
> HJT:=evalm(H&*transpose(J));
```

$$HJT := \begin{bmatrix} -2.000058823 \\ -.3\,10^{-9} \end{bmatrix}$$

```
> HJT[2,1]:=0; print(HJT);
```

$$HJT_{2,1} := 0$$

$$\begin{bmatrix} -2.000058823 \\ 0 \end{bmatrix}$$

```
> w:= vector([-h/HJT[1,1], 0]);#w = Hv
```

$$w := [.00002941013500, 0]$$

```
> v:=evalm(H&*w); # v^4
```

$$v := [-.00001315321114, -.00002630492499]$$

```
> L:= evalm(H&*E); BL:=evalm(V&*Phi&*L); # L_4 & (BL)_4
```

$$L := \begin{bmatrix} -.8944170093 \\ .4472339590 \end{bmatrix}$$

$$BL := \begin{bmatrix} -3.279562973 \\ .4944357453 \end{bmatrix}$$

```
> p:=matadd(phi, Phi&*v);#auxiliary variable
```

$$p := [.4472339593, .8944170095]$$

```
> r:= evalm(-V&*p);#RHS of overdetermined system to compute u in ODA
```

$$r := [-.149145866, -.9888152091]$$

```
> u:= leastsqrs(BL, r); # u^4
```

$$u := [.00002069770909]$$

```
> Deltax:= matadd(v, L&*u); # Deltax^4
```

$$Deltax := [-.00003166559420, -.00001704820661]$$

Fourth iteration is complete. Update $\mathbf{x}$:

```
> x:= matadd(x, Deltax);  x4:= evalm(x);# x^4
```

$$x := [.4472154469, .8944262662]$$

$$x4 := [.4472154469, .8944262662]$$

```
> f:= obj(x); #f_4
```

$$f := .5000000006$$

Given the norm of Deltax, we declare convergence here, and plot the iteration history in $x[1]$-$x[2]$ plane:

```
> o0:= evalm(x0); o1:= evalm(x1); o2:= evalm(x2); o3:= evalm(x3); o4:=
> evalm(x4);
```

$$o0 := [2, 2]$$

$$o1 := [.787500000, 1.462500000]$$

$$o2 := [.5090179236, 1.011062998]$$

$$o3 := [.4508808673, .9011974282]$$

$$o4 := [.4472154469, .8944262662]$$

```
> p0:=point(convert(o0,list), symbol=circle, color=blue);
> p1:=point(convert(o1,list), symbol=circle, color=blue);
> p2:=point(convert(o2,list), symbol=circle, color=blue);
> p3:=point(convert(o3,list), symbol=circle, color=blue);
> p4:=point(convert(o4,list), symbol=circle, color=blue);
```

$p0 :=$ POINTS([2., 2.], COLOUR(RGB, 0., 0., 1.00000000), SYMBOL($CIRCLE$))

$p1 :=$ POINTS([.787500000, 1.462500000], COLOUR(RGB, 0., 0., 1.00000000), SYMBOL($CIRCLE$))

$p2 :=$ POINTS([.5090179236, 1.011062998], COLOUR(RGB, 0., 0., 1.00000000), SYMBOL($CIRCLE$))

$p3 :=$ POINTS([.4508808673, .9011974282], COLOUR(RGB, 0., 0., 1.00000000), SYMBOL($CIRCLE$))

$p4 :=$ POINTS([.4472154469, .8944262662], COLOUR(RGB, 0., 0., 1.00000000), SYMBOL($CIRCLE$))

```
> l1 := arrow(convert(o0,list),convert(o1,list), 10.0, 0.1, .1, arrow,
> color=red, thickness=2):
> l2 := arrow(convert(o1,list),convert(o2,list), 6.0, 0.1, .2, arrow,
> color=green, thickness=2):
> l3 := arrow(convert(o2,list),convert(o3,list), 6.0, 0.1, .7, arrow,
> color=red, thickness=2):
> l4 := arrow(convert(o3,list),convert(o4,list), 6.0, 0.1, 6.0, arrow,
> color=green, thickness=2):
> c1 := arc([0,0], 1,-Pi/6..4*Pi/6,color=black, thickness=2):
> obj_plot:= proc(ax,ay) (1/2)*(9*ax^2 - 8*ax*ay + 3*ay^2)
> end:f1:=implicitplot(obj_plot-3,-0.5..2,-0.5..2.5,numpoints=3000,
> linestyle=4,color=blue):
> f2:=implicitplot(obj_plot-2,-0.5..2,-0.5..2,numpoints=3000,
> linestyle=4,color=blue):f3:=implicitplot(obj_plot-1,-0.5..2,-0.5..2,nu
> mpoints=6000,
> linestyle=4,color=blue):f4:=implicitplot(obj_plot-0.5,-0.5..2,-0.5..2,
> numpoints=6000, linestyle=4,color=blue):

> display({c1,p0,p1, p2, p3, p4, p1, l1, l2, l3, l4,f1,f2,f3,f4},
> insequence = false, color=red, scaling=constrained);
```

The plots produced by the plotting commands in the Maple worksheet are reproduced in Fig. 4.1.

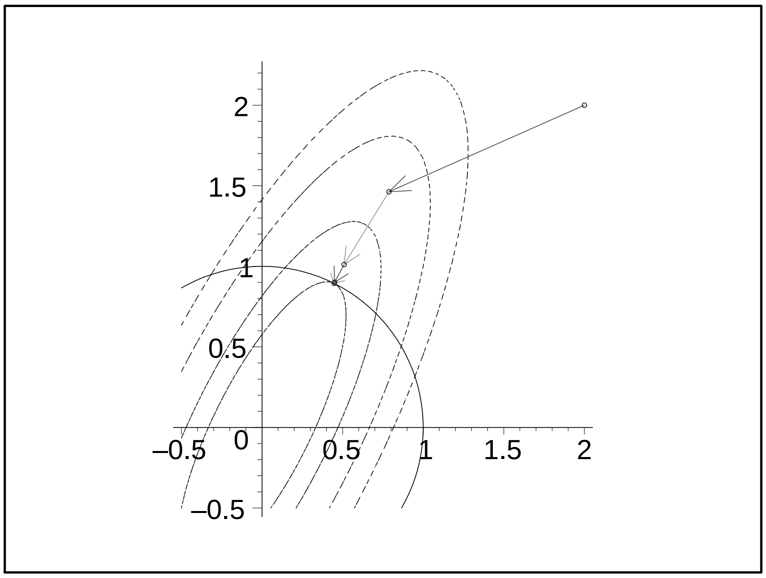

Figure 4.1: The four iterations leading to the solution of the linear least-square problem subject to one quadratic constraint

A plot of the contours of the objective function and the constraint, showing all four stationary points, is displayed in Fig. 4.2.

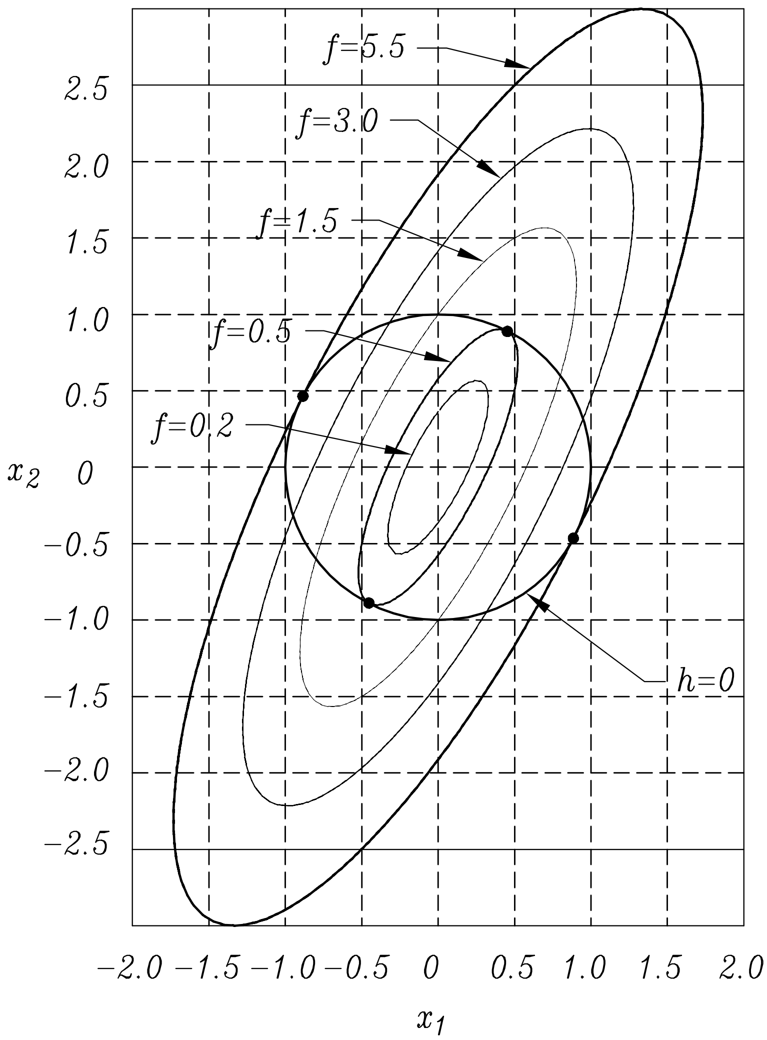

Figure 4.2: The contours of constant f and the constraint $h = 0$

Example 4.3.2 (Finding the Eigenvalues and Eigenvectors of a Symmetric Matrix)

The problem of finding the eigenvalues and corresponding eigenvectors of a symmetric $n \times n$ matrix $\mathbf{M}$ is solved as a linear least-square problem subject to quadratic constraints: For $i = 1, 2, \cdots, n$, and $k = 1, 2, \cdots, i$, find λ_i and $\mathbf{x}_i$ such that

$$\lambda_i = \min_{\mathbf{x}_i} \frac{1}{2}\mathbf{x}_i^T \mathbf{M}\mathbf{x}_i$$

subject to

$$\mathbf{x}_k^T \mathbf{x}_i = \begin{cases} 0, & \text{if } k = 1,\ 2,\ \cdots,\ i-1; \\ 1, & \text{if } k = i. \end{cases}$$

where λ_i *is the ith eigenvalue of matrix* $\mathbf{M}$ *and* $\mathbf{x}_i$ *is the corresponding eigenvector. In order to use the ODA package to solve the problem, we define, for* $i = 1, 2, \cdots, n$*:*

$$\begin{aligned} q &= n \qquad \text{and} \qquad l = i, \\ \mathbf{x} &= \mathbf{x}_i, \\ \boldsymbol{\phi}(\mathbf{x}) &= \mathbf{x}, \\ \mathbf{h}(\mathbf{x}) &= [\, \mathbf{x}_1^T \mathbf{x} \quad \cdots \quad \mathbf{x}_{i-1}^T \mathbf{x} \quad \mathbf{x}^T \mathbf{x} - 1 \,]^T, \\ \mathbf{W} &= \mathbf{M} \end{aligned} \tag{4.31}$$

where $\mathbf{x}_k$*, for* $k = 1, 2, \cdots, i-1$*, are the previously calculated eigenvectors of* $\mathbf{M}$*, and hence, are known. With the above definitions, for* $i = 1, 2, \cdots, n$*, subroutine* `LSSCNL` *of the ODA package is called n times. After each call, one eigenvalue and its corresponding eigenvector are obtained. Notice that, in the last call, the number of constraints is equal to the number of variables, namely,* $l = n$*. Matrix* $\mathbf{M}$ *is given as*

$$\mathbf{M} = \begin{bmatrix} 4 & 2 & 1 & 1 & 1 \\ 2 & 4 & 2 & 1 & 1 \\ 1 & 2 & 4 & 2 & 1 \\ 1 & 1 & 2 & 4 & 2 \\ 1 & 1 & 1 & 2 & 4 \end{bmatrix}.$$

We use the initial guess

$$\mathbf{x}^0 = [\, 0.1 \quad 0.1 \quad 0.1 \quad 0.1 \quad 0.1 \,]^T .$$

The eigenvalues and the eigenvectors computed with the ODA package are listed in Table 4.1. In that table, the number of iterations that the package took till convergence was reached with $\epsilon_1 = 0.0001$ *and* $\epsilon_2 = 0.0001$*, is indicated.*

4.4 Equality-Constrained Optimization with Arbitrary Objective Function

The problem to be solved is defined as:

$$f = f(\mathbf{x}) \quad \rightarrow \quad \min_{\mathbf{x}} \tag{4.32}$$

i	1	2	3	4	5
λ_i	1.27738	3.08749	9.63513	2.0	4.0
1st comp. of $\mathbf{x}_i$	0.26565	−0.51369	0.40689	0.5	0.5
2nd comp. of $\mathbf{x}_i$	−0.51853	0.10368	0.46944	−0.5	0.5
3rd comp. of $\mathbf{x}_i$	0.56667	0.67138	0.47764	0.0	0.0
4th comp. of $\mathbf{x}_i$	−0.51853	0.10368	0.46944	0.5	−0.5
5th comp. of $\mathbf{x}_i$	0.26565	−0.51369	0.40689	−0.5	−0.5
# of iterations	12	11	6	38	54

Table 4.1: Eigenvalues and eigenvectors of $\mathbf{M}$

subject to the nonlinear equality constraints

$$\mathbf{h}(\mathbf{x}) = \mathbf{0} \tag{4.33}$$

where $\mathbf{x}$ is the n-dimensional design-variable vector, the objective function $f(\mathbf{x})$ being a nonlinear function of $\mathbf{x}$, not necessarily quadratic in the sense of Section 4.3, but with continuous derivatives up to the second order. Moreover, $\mathbf{h}(\mathbf{x})$ is a l-dimensional vector of nonlinear equality constraints, with a continuous gradient.

In the problem defined in eq.(4.32), if the constraints in eq.(4.33) are *analytic*, then there exists a *feasible manifold* $\mathcal{F} \subset \mathbb{R}^n$, of dimension l, such that, if $\mathbf{u} \in \mathcal{F}$. In this case, under $\mathbf{x} = \mathbf{x}(\mathbf{u})$,

$$\mathbf{h}(\mathbf{x}(\mathbf{u})) = \mathbf{0} \tag{4.34}$$

In the particular case in which $\mathbf{h}(\mathbf{x}(\mathbf{u}))$ is linear, then $\mathcal{F}$ is a vector space, i.e., the *feasible space* of the problem at hand.

At $\mathbf{x} = \mathbf{x}^k$, we assume that, in general, $\mathbf{h}(\mathbf{x}^k) = \mathbf{h}^k \neq \mathbf{0}$, i.e., the current $\mathbf{x}$ is not feasible, and note that $f(\mathbf{x}^k + \Delta\mathbf{x}^k)$ can be expanded, to a second order, as

$$f(\mathbf{x}^k + \Delta\mathbf{x}^k) \approx f(\mathbf{x}^k) + (\nabla f)_k^T \Delta\mathbf{x}^k + \frac{1}{2}(\Delta\mathbf{x}^k)^T(\nabla\nabla f)_k \Delta\mathbf{x}^k \rightarrow \min_{\Delta\mathbf{x}^k} \tag{4.35a}$$

subject to

$$\mathbf{J}_k \Delta\mathbf{x}^k = -\mathbf{h}^k \tag{4.35b}$$

We have thus derived a linear least-square problem in $\Delta\mathbf{x}^k$ subject to the linear constraints (4.35b). To find the increment $\Delta\mathbf{x}^k$, we resort to the ODA, as introduced in Section 4.2. To this end, we decompose the foregoing vector into its two orthogonal components:

$$\Delta\mathbf{x}^k = \Delta\mathbf{x}_0^k + \mathbf{L}_k \Delta\mathbf{u}^k$$

where

$$\Delta \mathbf{x}_0^k = -\mathbf{J}_k^T(\mathbf{J}_k\mathbf{J}_k^T)^{-1}\mathbf{h}^k \tag{4.36}$$

is the minimum-norm solution of eq.(4.35b), and $\mathbf{L}_k$ is the isotropic orthogonal complement of $\mathbf{J}_k$ defined in eqs.(4.13) and (4.14), while $\mathbf{J}_k$ itself is defined as the gradient of $\mathbf{h}$ with respect to $\mathbf{x}$, evaluated at $\mathbf{x} = \mathbf{x}^k$. Moreover, $\mathbf{L}_k$ and $\Delta\mathbf{u}^k$ are found with the procedure described in Section 4.2 for linearly-constrained linear least-square problems. Furthermore, with $\Delta\mathbf{x}_0^k$ given by eq.(4.36), $f(\mathbf{x}^k + \Delta\mathbf{x}^k)$ becomes a function solely of $\Delta\mathbf{u}^k$, i.e.,

$$\begin{aligned} f(\Delta\mathbf{u}^k) \approx \tilde{f}(\Delta\mathbf{u}^k) \equiv\ & f(\mathbf{x}^k) + (\nabla f)_k^T(\Delta\mathbf{x}_0^k + \mathbf{L}_k\Delta\mathbf{u}^k) \\ & + \frac{1}{2}(\Delta\mathbf{x}_0^k + \mathbf{L}_k\Delta\mathbf{u}^k)^T(\nabla\nabla f)_k(\Delta\mathbf{x}_0^k + \mathbf{L}_k\Delta\mathbf{u}^k) \rightarrow \min_{\Delta\mathbf{u}^k} \end{aligned}$$

which can be cast in the form

$$\begin{aligned} \tilde{f}(\Delta\mathbf{u}^k) =\ & \frac{1}{2}(\Delta\mathbf{u}^k)^T\mathbf{L}_k^T(\nabla\nabla f)_k\mathbf{L}_k\Delta\mathbf{u}^k + \left[\mathbf{L}_k^T(\nabla\nabla f)_k\Delta\mathbf{x}_0^k + \mathbf{L}_k^T(\nabla f)_k\right]^T\Delta\mathbf{u}^k \\ & + \frac{1}{2}(\Delta\mathbf{x}_0^k)^T(\nabla\nabla f)_k\Delta\mathbf{x}_0^k + (\nabla f)_0^T\Delta\mathbf{x}_0^k + f(\mathbf{x}^k) \rightarrow \min_{\Delta\mathbf{u}^k} \end{aligned} \tag{4.37}$$

subject to *no constraints*, $\tilde{f}(\Delta\mathbf{u}^k)$ being *quadratic* in $\Delta\mathbf{u}^k$. Function $\tilde{f}(\Delta\mathbf{u}^k)$ has a minimum if its Hessian with respect to $\Delta\mathbf{u}^k$, $\mathbf{H}_k = \mathbf{L}_k^T(\nabla\nabla f)_k\mathbf{L}_k$, is positive-definite. Under the assumption that this is the case, then, the minimum $\Delta\mathbf{u}^k$ of $\tilde{f}(\Delta\mathbf{u}^k)$ can be readily computed upon zeroing its gradient with respect to $\Delta\mathbf{u}^k$, which yields

$$\mathbf{H}_k\Delta\mathbf{u}^k = -\mathbf{L}_k^T(\nabla\nabla f)_k\Delta\mathbf{x}_0^k - \mathbf{L}_k^T(\nabla f)_k$$

Under the assumption that $\mathbf{H}_k$ is positive-definite, it is invertible, and hence,

$$\Delta\mathbf{u}^k = -\mathbf{H}_k^{-1}[\mathbf{L}_k^T(\nabla\nabla f)_k\Delta\mathbf{x}_0^k + \mathbf{L}_k^T(\nabla f)_k] \tag{4.38}$$

We have thus reduced the original problem to a *sequence* of *linear-quadratic programs*. This means that we have solved the problem iteratively. At each iteration, moreover, we find the correction to the current approximation $\Delta\mathbf{x}^k$ by means of a combination of two linear problems, one being a minimum-norm problem, the other involving a determined linear system of equations. For this reason, the above procedure is called *sequential quadratic programming*.

The foregoing procedure relies on the rather daring assumption that the Hessian $\mathbf{H}_k$ is positive-definite. Below we study the more realistic case of a non-positive-definite Hessian.

4.4.1 Sequential Quadratic Programming with Hessian Stabilization

In the presence of a non-positive-definite Hessian $\mathbf{H}_k$, we aim at a *perturbation* $\Delta\mathbf{H}_k$ of the Hessian that will render the *perturbed Hessian* $\tilde{\mathbf{H}}_k$ positive-definite, thus producing

$$\tilde{\mathbf{H}}_k \equiv \mathbf{H}_k + \Delta\mathbf{H}_k \tag{4.39}$$

How to obtain $\Delta\mathbf{H}_k$ that is guaranteed to produce a positive-definite-Hessian is the key issue here. Various methods are available to do this. Most of these methods aim at producing not a positive-definite Hessian itself, but rather an estimate of its inverse which is what we actually need. The most favoured method in this respect is that of Broyden, Fletcher, Goldfarb, and Shanno, known as the BFGS method. Essentially, this method is applied to the unconstrained minimization of nonlinear objective functions, based on the system of nonlinear equations derived from the normality conditions. The Jacobian of this system is nothing but the Hessian of the objective function.

We describe in the subsection below a method for the determination of $\Delta\mathbf{H}_k$ introduced in eq.(4.39). Note that, once the perturbed Hessian, which is most frequently referred to as the *stabilized Hessian*, is available, $\Delta\mathbf{u}^k$ is found from

$$\tilde{\mathbf{H}}_k\Delta\mathbf{u}^k = -\mathbf{L}_k^T[(\nabla\nabla f)_k\Delta\mathbf{x}_0^k + (\nabla f)_k] \tag{4.40}$$

The process of finding a positive-definite $\tilde{\mathbf{H}}_k$ is termed *Hessian stabilization*. The rationale behind Hessian stabilization lies in the property that, if the eigenvalues of a $n \times n$ matrix $\mathbf{M}$ are $\{\,\mu_k\,\}_1^n$, then the eigenvalues of matrix $\mathbf{M} + \alpha\mathbf{1}$, where α is a real number and $\mathbf{1}$ is the $n \times n$ identity matrix, are $\{\,\mu_k + \alpha\,\}_1^n$. Thus, the effect of adding the isotropic matrix $\alpha\mathbf{1}$ to $\mathbf{M}$ is to shift the eigenvalues of the latter to the right of the complex plane by an amount α if $\alpha > 0$; if $\alpha < 0$, then the same isotropic matrix shifts the eigenvalues of $\mathbf{M}$ to the left of the complex plane by an amount $|\alpha|$. If the Hessian of interest is not positive-definite, this means that it has some negative eigenvalues, Hessian stabilization thus consisting in finding the right value of α in the foregoing scheme that will shift the Hessian eigenvalues to the right of the real axis—since the Hessian is necessarily symmetric, its eigenvalues are all real—so that none of the shifted eigenvalues will lie on the left half of the real axis. Notice that, if $\alpha > 0$ is underestimated, then the associated isotropic matrix will fail to shift some of the negative Hessian eigenvalues to the right; if overestimated, then all shifted eigenvalues will lie on the right half of the real axis, but the Hessian will be overly perturbed, and the converegence will slow down.

Obviously, if we know the eigenvalues of the Hessian $\mathbf{H}_k$, then we can find the right μ_k that will shift all its eigenvalues to the right. However, computing eigenvalues is an iterative process, except for very special cases of simple matrices, and hence, we cannot relie on knowledge of those eigenvalues. We discuss below how to estimate the right amount of shift α without having to compute the Hessian eigenvalues. We do this by means of an estimate of the location of the Hessian eigenvalues, as provided by the *Gerschgorin Theorem.*

4.4.2 The Gerschgorin Theorem and Its Application to Hessian Stabilization

The *Gerschgorin Theorem* establishes a region in the complex plane containing all the eigenvalues of a $n \times n$ matrix $\mathbf{A}$, defined over the complex field $\mathbb{C}$, namely,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

According to the Gerschgorin Theorem, all the eigenvalues of $\mathbf{A}$ lie within a complex region $\mathcal{S}$, defined as the union of disks $\mathcal{D}_i$ centered at a_{ii}, with radius r_i, in the complex plane, for $i = 1, ..., n$, r_i being given by

$$r_i = \sum_{j=1, j\neq i}^{n} \mid a_{ij} \mid$$

in which $|\cdot|$ denotes the module of $(\cdot)$. The Gerschgorin Theorem is illustrated in Fig. 4.3, the region $\mathcal{S}$ thus being

$$\mathcal{S} = \bigcup_{i=1}^{n} \mathcal{D}_i$$

If $\mathbf{A}$ is symmetric and real, which is so for Hessian matrices, then its eigenvalues lie in the union of the real intervals

$$I_i = [a_{ii} - r_i, a_{ii} + r_i], \quad i = 1, 2, ..., n$$

A lower bound l of the set $\{\lambda_i\}_1^n$ of eigenvalues of $\mathbf{A}$ is, thus,

$$l \equiv \min_i \{a_{ii} - r_i\}_1^n \tag{4.41a}$$

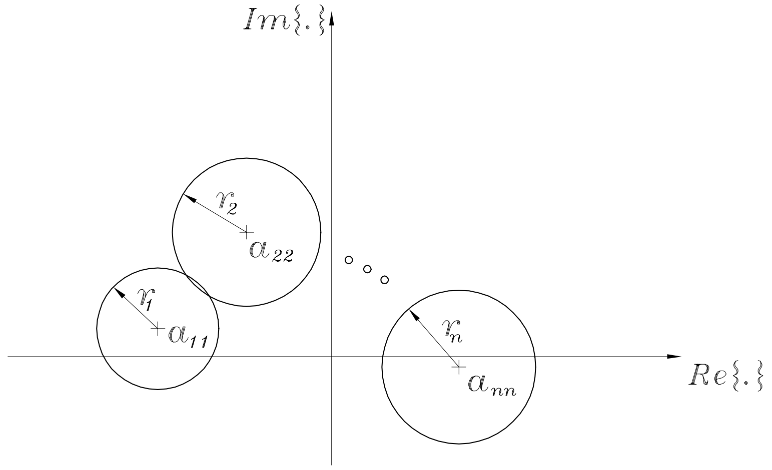

Figure 4.3: The Gerschgorin disks.

the corresponding upper bound being

$$u \equiv \max_i \{a_{ii} + r_i\}_1^n \tag{4.41b}$$

If $\mathbf{A}$ is positive-definite, all eigenvalues of $\mathbf{A}$ must be positive, which means that the lower bound l should be positive as well. If, on the other hand, $\mathbf{A}$ is either sign-indefinite or positive-definite, but close to singular, then l can be negative.

4.4.3 Hessian Stabilization with the Aid of the Gerschgorin Theorem

The feasible Hessian matrix $\mathbf{L}_k^T(\nabla\nabla f)|_k\mathbf{L}_k$ of the objective function fails to be positive-definite when the Hessian $(\nabla\nabla f)|_k$ fails, in turn, to be positive-definite. However, it may well happen that the latter fail to be positive-definite and yet the former be positive-definite. In this light, it appears that we need not stabilize the Hessian itself, but only its feasible projection. In practice, we have found that stabilizing the Hessian, rather than only its feasible projection, leads to a more robust procedure. We will thus proceed accordingly.

The stabilizing procedure is applied by introducing a scalar $\mu_k > -l$, such that a new positive-definite matrix $\mathbf{W}_k$ is used to replace $(\nabla\nabla f)_k$, with $\mathbf{W}_k$ defined as

$$\mathbf{W}_k = (\nabla\nabla f)_k + \mu_k \mathbf{1} \tag{4.42}$$

$\mathbf{W}_k$ thus being guaranteed to be positive-definite, for its minimum eigenvalue is *a fortiori* greater than zero.

The stabilized Hessian thus yields the feasible Hessian

$$\tilde{\mathbf{H}}_k = \mathbf{L}_k^T \mathbf{W}_k \mathbf{L}_k = \mathbf{L}_k^T (\nabla\nabla f)|_k \mathbf{L}_k + \mu_k \mathbf{L}_k^T \mathbf{L}_k \tag{4.43}$$

Moreover, since $\mathbf{L}_k$ is isotropic, it turns out that

$$\mathbf{L}_k^T \mathbf{L}_k = \mathbf{1}_{n'} \tag{4.44}$$

with $\mathbf{1}_{n'}$ denoting the $(n-l) \times (n-l)$ identity matrix. Therefore, the *stabilizied feasible Hessian* $\tilde{\mathbf{H}}_k$ reduces to

$$\tilde{\mathbf{H}}_k = \mathbf{L}_k^T \mathbf{W}_k \mathbf{L}_k = \mathbf{L}_k^T (\nabla\nabla f)|_k \mathbf{L}_k + \mu_k \mathbf{1}_{n'} \tag{4.45}$$

which is now positive-definite, problem (4.37) thus admitting one minimum $\Delta\mathbf{u}^k$, which is computed from eq.(4.40).

Choice of the Gerschgorin Shift

In this subsection we stress the importance of the selection of μ. With a proper selection, the number of iterations can be effectively reduced.

The selection of μ is suggested to be *slightly* greater than the lower bound l of the eigenvalues, as obtained by the Gerschgorin Theorem, i.e.,

$$\mu = -(1 + \Delta) l \tag{4.46}$$

where Δ is a positive number, that is to be chosen as small as possible.

The value of Δ is related to the bandwidth b of the Hessian eigenvalues, with b defined as

$$b = u - l$$

Example 4.4.1 (Powell's Function)
A problem proposed by Powell (1969) is solved here:

$$f(\mathbf{x}) = e^{x_1 x_2 x_3 x_4 x_5} \quad \rightarrow \quad \min_{\mathbf{x}}, \quad \mathbf{x} \equiv [\, x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \,]^T$$

subject to the nonlinear equality constraints

$$\begin{aligned} h_1 &= x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ h_2 &= x_2 x_3 + x_4 x_5 = 0 \\ h_3 &= x_1^3 + x_2^3 + 1 = 0 \end{aligned}$$

A word of caution is in order here: while the exponential function e^x *is convex—its second derivative with respect to* x *is positive everywhere—the bivariate exponential function* $\mathbf{e}^{x_1 x_2}$ *is not convex everywhere, and neither is so the above objective function. In fact, the Hessian of the bivariate exponential becomes sign-indefinite in a region of the* x_1*-*x_2 *plane. This statement is illustrated with the plot of this function displayed in Fig. 4.4. The conclusion of the foregoing discussion is, then, that the multivariable exponential function, like Powell's function, has a Hessian that is sign-indefinite in a region of* $\mathbb{R}^5$*. Optimum solutions were obtained with two different algorithms, the corresponding results being listed in Table 4.2. Results were obtained under the same environment, a Silicon Graphics 64-bit Octane SE workstation, with a 250 MHz R10000 processor, running the IRIX 6.5 operating system. An initial guess is taken as*

$$\mathbf{x}_0 = [\,-1 \quad 2 \quad -0.5 \quad 1 \quad 2\,]^T$$

with tolerance of 10^{-6}*. The ODA package requires only 58 iterations, as compared with 186 required by the Matlab Optimization Toolbox. Moreover, the CPU time required by the ODA is only 8.9 % of the CPU time consumed by Matlab.*

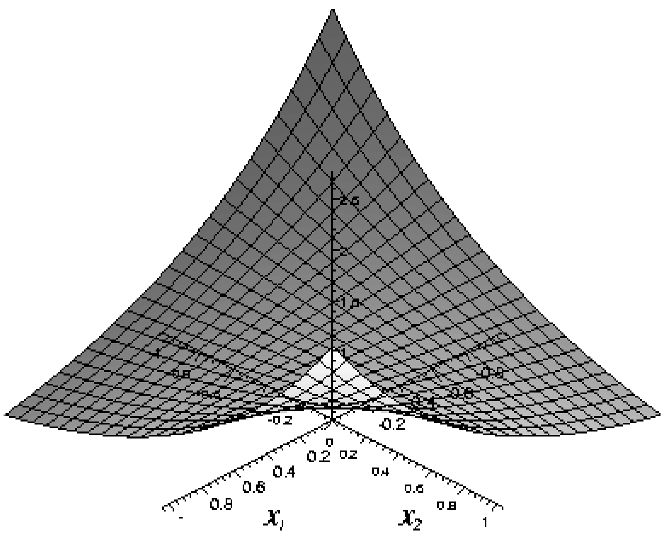

Figure 4.4: The bivariate exponential $e^{x_1 x_2}$

Example 4.4.2 (The Equilibrium Configuration of an N-link Chain) *Shown in Fig. 4.5a is a chain with N links in its equilibrium configuration, which*

Table 4.2: A performance comparison based on Powell's function

	Matlab	ODA
f	0.05395	0.05395
x_1	-1.7172	-1.7171
x_2	1.5957	1.5957
x_3	1.8272	-1.8272
x_4	0.7636	-0.7636
x_5	0.7636	0.7636
# of iterations	186	55
CPU time (s)	0.2903	0.0259

spans a distance d, with each link of length ℓ. Knowing that the chain reaches its equilibrium configuration when its potential energy attains a minimum value, find the said equilibrium configuration. This problem, originally proposed by Luenberger (1984), was solved for the case of two design variable, exactly, in Example 3.3.2.

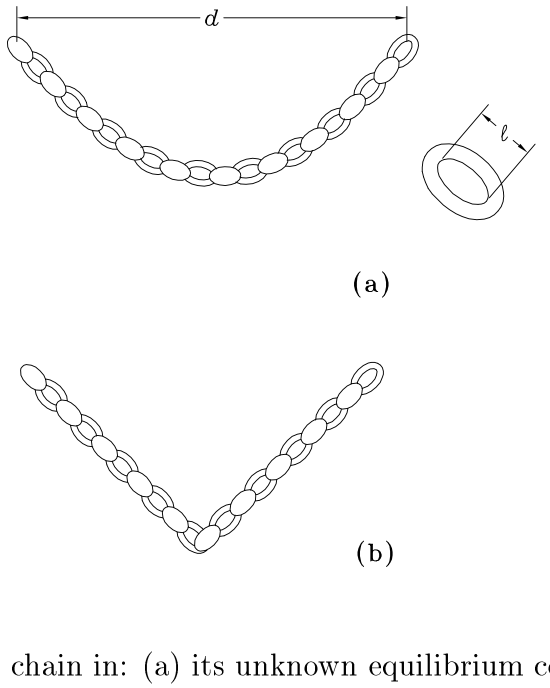

Figure 4.5: An N-link chain in: (a) its unknown equilibrium configuration; and (b) a configuration to be used as an initial guess

Angles θ_i, used to define the configuration of the chain, are measured from the vertical ccw, with θ_i corresponding to the angle that the axis of the ith link makes *with the vertical, as shown in Fig. 4.6.*

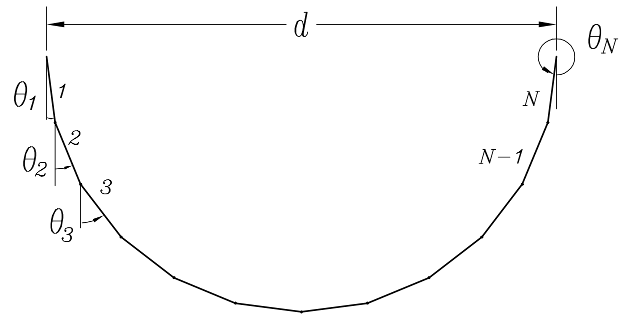

Figure 4.6: Definition of θ_i for the N-link chain

If $V \equiv \mu\ell f(\theta_1, \theta_2 \ldots, \theta_N)$ *denotes the potential energy of the chain, and* μ *is the mass density of the links per unit length, then minimizing* V *is equivalent to minimizing* f*, which is given by*

$$f(\theta_1, \theta_2 \ldots, \theta_N) = -\left[\frac{1}{2}\cos\theta_1 + (\cos\theta_1 + \frac{1}{2}\cos\theta_2) + \ldots \right.$$
$$\left. + (\cos\theta_1 + \ldots + \cos\theta_{N-1} + \frac{1}{2}\cos\theta_{N-1})\right] \quad \rightarrow \quad \min_{\{\theta_i\}_1^N}$$

or, in compact form,

$$f(\theta_1, \theta_2 \ldots, \theta_N) = -\frac{1}{2}\sum_{i=1}^{N}[2(N-i)+1]\cos\theta_i \quad \rightarrow \quad \min_{\{\theta_i\}_1^N}$$

subject to two constraints: the two ends (1) must lie at the same height, and (2) are separated by a distance d, as shown in Fig. 4.6. The constraints are

$$h_1 = \sum_{i=1}^{N}\cos\theta_i = 0$$
$$h_2 = \sum_{i=1}^{N}\sin\theta_i - \frac{d}{\ell} = 0$$

Under the assumption that the configuration is symmetric, and that N *is even, then* $M = N/2$ *is an integer. Thus, only one half of the chain need be considered. The problem is, thus, simplified as*

Table 4.3: Luenberger's chain with $M = 5$

	Matlab	The ODA
θ_1	0.0893	0.0893
θ_2	0.1147	0.1147
θ_3	0.1599	0.1599
θ_4	0.2625	0.2625
θ_5	0.67856	0.6785
f_{min}	-12.2650	-12.2650
Iterations	16	7
CPU time (s)	0.2825	0.003261

$$
\begin{aligned}
f(\theta_1, \theta_2 \ldots, \theta_M) = & - \left[\frac{1}{2}\cos\theta_1 + (\cos\theta_1 + \frac{1}{2}\cos\theta_2) + \ldots \right. \\
& \left. + (\cos\theta_1 + \ldots + \cos\theta_{M-1} + \frac{1}{2}\cos\theta_M) \right] \\
= & -\frac{1}{2}\sum_{i=1}^{M}[2(M-i)+1]\cos\theta_i \quad \rightarrow \quad \min_{\{\cos\theta_i\}_1^M}
\end{aligned}
$$

The two constraints then reduce to only one:

$$
h = \sum_{i=1}^{M} \sin\theta_i - \frac{d}{2\ell} = 0
$$

This problem, with $M = 5$, i.e., with $N = 10$, is solved now using the configuration of Fig. 4.5b as an initial guess. The equilibrium configuration of the chain is given in Table 4.3 with a comparison between ODA and Matlab.

With the same tolerance set at 0.0001, the ODA takes less than half the number of iterations than Matlab; additionally, the CPU time consumed by ODA is about 10% of that consumed by Matlab.

It is noteworthy that the convergence of ODA is dependent on the choice of μ_k in eq.(4.42), for a given value of $d/(2l)$.

Example 4.4.3 (Minimum value of the Rosenbrock Function) *We include an example where the ODA package is used to find the minimum value of the Rosenbrock function (Rosenbrock, 1960), a.k.a. the* banana function, *defined as*

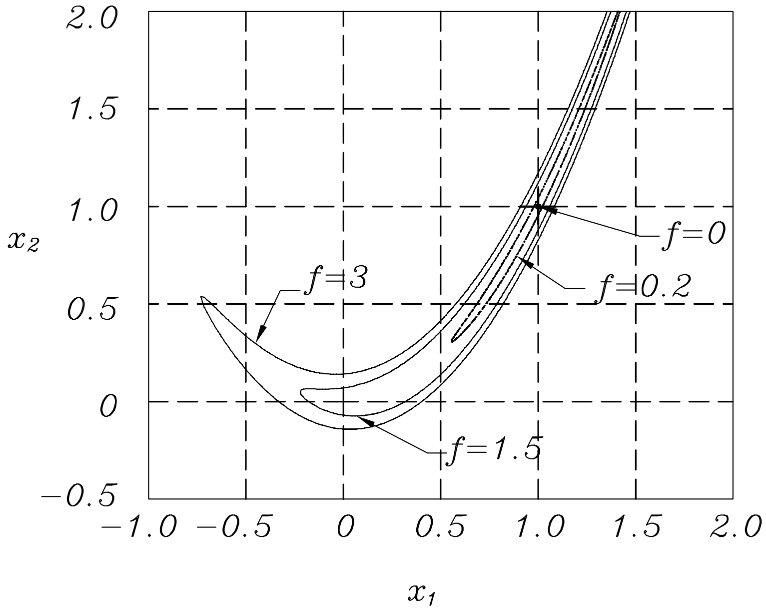

Figure 4.7: The contours of the Rosenbrock (a.k.a. the banana) function

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{4.47}$$

The problem can be treated as finding an "approximate" solution of the following system of nonlinear equations

$$\boldsymbol{\phi} = \begin{bmatrix} x_2 - x_1^2 \\ 1 - x_1 \end{bmatrix} \tag{4.48}$$

such that the least-square error f is a minimum, i.e.,

$$f(\mathbf{x}) = \frac{1}{2}\boldsymbol{\phi}^T \mathbf{W} \boldsymbol{\phi} \quad \to \quad \min_{x_1, x_2} \tag{4.49}$$

with

$$\mathbf{W} = \begin{bmatrix} 200.0 & 0 \\ 0 & 2.0 \end{bmatrix} \tag{4.50}$$

By taking $\mathbf{x}_0 = [0.2 \quad 0.2]^T$ as initial guess, we obtained the sequence of values $\mathbf{x}^1, \mathbf{x}^2, \ldots$ shown in Table 4.4. The optimum was reached after 3 iterations, and found to be $\mathbf{x}_{opt} = [1 \quad 1]^T$. The contours of the banana function are plotted in Fig. 4.7. Now, since $\boldsymbol{\phi} = \mathbf{0}$ at $\mathbf{x}_{opt}$, the normality condition (2.77) is readily verified.

Table 4.4: Interations toward the minimum of the Rosenbrock function

i	1	2	3
$\mathbf{x}^i$	$[1.000000,\ 0.360000]^T$	$[1.000000,\ 1.000000]^T$	$[1.000000,\ 1.000000]^T$

Example 4.4.4 (The Constrained Minimiation of the Rosenbrock Function) *In this example, we find its equality-constrained minimum of the Rosenbrock function using SQP via the ODA. We thus have*

$$f(x_1,\, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad \rightarrow \quad \min_{x_1, x_2}$$

subject to

$$\begin{aligned} h((x_1,\, x_2) = {} & 0.7525x_1^2 - 1.1202x_1 - 0.8574x_1x_2 \\ & +0.6168x_2 + 0.2575x_2^2 + 0.4053 \end{aligned}$$

The function is notorious for its ill-conditioning, which is apparent from its contours, as shown in Fig. 4.8, showing elongated valleys. The outcome is that the quadratic approximation of this function within those valleys is a family of ellipses that have one semiaxis much greater than the other one, thereby leading to ill-conditioning. Notice that the constraint is a rather elongated ellipse that contributes to the ill-conditioning of the problem.

Starting from the initial guess $\mathbf{x} = [1.5\ 1.5]$ *and damping ratio being 0.025, the optimum solution is found, in 312 ODA iterations, the results being*

$$\mathbf{x}_{opt} = \begin{bmatrix} 0.9176 \\ 0.5873 \end{bmatrix}$$

which yields

$$f_{\min} = 6.6963$$

4.5 Appendix A: The Damping Factor

This Appendix is included for completeness. It complements rather Ch. 3.

When implementing the Newton-Gauss method, the objective function f may increase upon correcting $\mathbf{x}^k$ according to eq.(2.70), i.e.

$$f(\mathbf{x}^{k+1}) > f(\mathbf{x}^k) \tag{4.51}$$

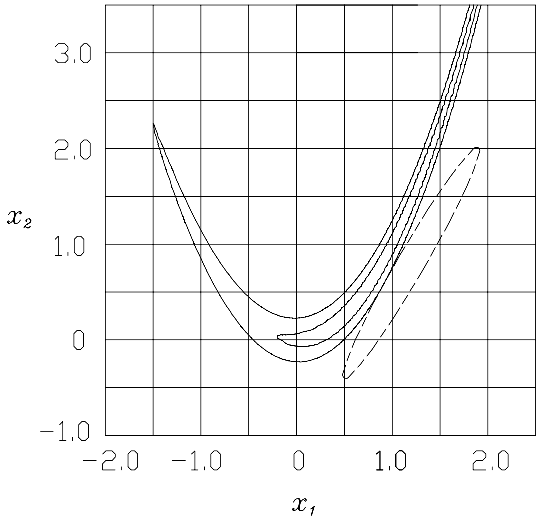

Figure 4.8: The contours of the Rosenbrock (banana) function and its quadratic constraint (dashed)

This increase gives rise to oscillations and sometimes even leads to divergence. One way to cope with this situation is by introducing *damping*. Instead of using the whole increment $\Delta \mathbf{x}^k$, we use a fraction of it, i.e.

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha \Delta \mathbf{x}^k, \quad 0 < \alpha < 1 \tag{4.52}$$

where α is known as the *damping factor*.

4.6 Appendix B: Computer Implementation Using ODA—A C-Library of Routines for Optimum Design

ODA is a C library of subroutines for optimization problems. The source file of this package, implemented in C, consists of a number of subroutines designed and classified based on their application. At the beginning of each subroutine a detailed description of the purpose and usage of the subroutine is included. Moreover, data validation has been considered in the software. In order to solve a problem, the user simply calls one corresponding C subroutine.

Since the solutions for linear problems are *direct*—as opposed to *iterative*—the use of ODA to solve linear problems requires only information on the problem parameters, such as matrices $\mathbf{A}$, $\mathbf{C}$, and $\mathbf{W}$, as well as vectors $\mathbf{b}$ and $\mathbf{d}$, as applicable. For nonlinear problems, the solution is iterative, and hence, the user is required to provide functions describing $\boldsymbol{\phi}(\mathbf{x})$, $\mathbf{h}(\mathbf{x})$, $\mathbf{\Phi}(\mathbf{x})$, $\mathbf{J}(\mathbf{x})$, as needed. These functions are provided via subroutines in forms that can be called by the package. In addition to this information, the user is also required to provide an initial guess $\mathbf{x}_0$ of $\mathbf{x}$, so that the iterative procedure can be started.

1. **Unconstrained linear problems:** Subroutine `MNSLS` is used to find the minimum-norm solution of an underdetermined linear system, while subroutine `LSSLS` is used to find the least-square approximation of an overdetermined linear system. `LSSLS` can also handle determined systems, i.e., systems of as many equations as unknowns.

2. **Unconstrained nonlinear problems:** Subroutine `LSSNLS` is used to solve this type of problems. Since the nonlinear functions and their associated gradient matrices are problem-dependent, the user is required to provide two subroutines that are used to evaluate the foregoing items, namely,

   - `FUNPHI`: This subroutine is used to evaluate the q-dimensional vector function $\boldsymbol{\phi}(\mathbf{x})$ in terms of the given n-dimensional vector $\mathbf{x}$.

   - `DPHIDX`: This subroutine is used to evaluate the $q \times n$ gradient matrix $\mathbf{\Phi}$ of the vector-function $\boldsymbol{\phi}(\mathbf{x})$ with respect to $\mathbf{x}$, at the current value of $\mathbf{x}$. Moreover an initial guess of $\mathbf{x}$ is required when calling this subroutine.

3. **Constrained linear problems:** Subroutine `LSSCLS` is used to solve this type of problems.

4. **Constrained nonlinear problems:** Subroutine `LSSCNL` is used for solving this type of problems. Before calling `LSSCNL`, the user is required to provide four problem-dependent subroutines: Two of these are `FUNPHI` and `DPHIDX`, already derscribed in item 2 above. The other two are used to evaluate the left-hand sides of the constraint equations and their gradient matrix, as listed below:

   - `FUNH`: This subroutine is used to evaluate the l-dimensional constraint function $\mathbf{h}$ in terms of the given n-dimensional vector $\mathbf{x}$.

- `DHDX`: This subroutine is used to evaluate the $l \times n$ gradient matrix $\mathbf{J}$ of the vector-function $\mathbf{h}(\mathbf{x})$ in terms of the given n-dimensional vector $\mathbf{x}$. Moreover, an initial guess of $\mathbf{x}$ is required when calling `LSSCNL`.

5. **Constrained problems with arbitrary objective function:** Subroutine `ARBITRARY` is used for solving this type of problems. Before calling `ARBITRARY`, the user is required to provide four problem-dependent subroutines: Two of these are `FUNPHI` and `DPHIDX`, as described in item 2 above. The other two subroutines are used to evaluate the left-hand sides of the constraint equations and their gradient matrix, as listed below:

   - `phi`: Subroutine used to evaluate the objective function $\boldsymbol{\phi}(\mathbf{x})$ in terms of the given n-dimensional vector $\mathbf{x}$.
   - `h`: Subroutine used to evaluate the l-dimensional constraint function $\mathbf{h}$ in terms of the given n-dimensional vector $\mathbf{x}$.
   - `J`: Subroutine used to evaluate the $l \times n$ gradient matrix $\mathbf{J}$ of the vector-function $\mathbf{h}(\mathbf{x})$ at the current value of $\mathbf{x}$.
   - `gradient`: Subroutine used to evaluate the n-dimensional gradient ∇f of the objective function $f(\mathbf{x})$ at the current value of vector $\mathbf{x}$.
   - `Hessian`: Subroutine used to evaluate the $n \times n$ Hessian matrix $\nabla\nabla f$ of the objective function $f(\mathbf{x})$ at the current value of vector $\mathbf{x}$. Moreover, an initial guess of $\mathbf{x}$ is required when calling `ARBITRARY`.

References

Bertsekas, D. P., 1995, *Nonlinear Programming*, Athena Scientific, Belmont, Massachusetts.

Broyden, C. G., 1970, "The convergence of a class of double-rank minimization algorithm", *J. Inst. Maths. Applications*, Vol. 6, pp. 76-90.

Fletcher, R., 1970, "A new approach to variable metric algorithms", *Computer Journal*, Vol. 13, pp. 317-322.

Goldfarb, D., 1970, "A family of variable metric updates derived by variational means", *Mathematics of Computing*, Vol. 24, pp. 23-26.

Lalee, M., Nocedal, J. and Plantenga, 1998, "On the implementation of an algorithm for large-scale equality constrained optimization", *SIAM J. Optim.*, Vol. 8, pp. 682-706.

Luenberger, D. G., 1984, *Linear and Nonlinear Programming*, Second Edition, Addision-Wesley Publishing Company, Reading, MA.

Murray, W., 1997, "Sequential quadratic programming methods for large problems", *Comput. Optim. Appl.*, Vol. 7, pp. 127-142.

Powell, M. J. D., 1969, "A method for Nonlinear Constraints in Minimizing Problems", in Fletcher, R. (Ed.), *Optmization*, Academic Press., N. Y., pp. 283-298.

Rao, S. S., 1996, *Engineering Optimization*, 3rd. Edition, John Wiley and Sons, Inc., New York.

Rosenbrock, H. H., 1960, "An automatic method for finding the greatest or the least value of a function", *Computer Journal*, Vol. 3. No. 3, pp. 175-184.

Shanno, D. F., 1970, "Conditioning of quasi-Newton methods for function minimization", *Mathematics of Computing*, Vol. 24, pp. 647-656.

Strang, G., 1988, *Linear Algebra*, Third Edition, Harcourt Brace Jovanovich College Publishers, Fort Worth.

Teng, C. P. and Angeles, J., 2001, "A sequential-quadratic-programming algorithm using orthogonal decomposition with Gerschgorin stabilization", *ASME J. of Mechanical Design*, Vol. 123, pp. 501-509.

Varga, R. S., 2000, *Matrix Iterative Analysis*, Prentice-Hall Inc., Englewood Cliffs, N. J..