Quasi-Riemannian Multiple Gradient Descent Algorithm for constrained multiobjective differential optimization

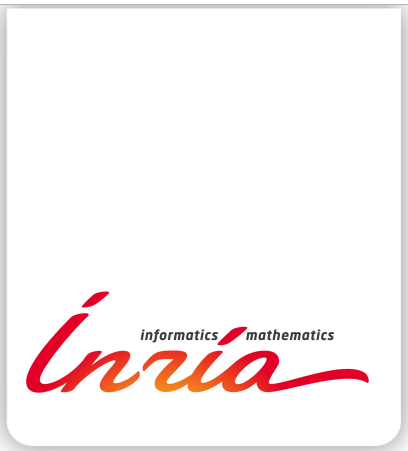
Jean-Antoine Désidéri

Quasi-Riemannian Multiple Gradient Descent Algorithm for constrained multiobjective differential optimization

Jean-Antoine Désidéri

RESEARCH REPORT
N° 9159
21 March 2018
Project-Team Acumes

ISSN 0249-6399 ISRN INRIA/RR--9159--FR+ENG



Quasi-Riemannian Multiple Gradient Descent Algorithm for constrained multiobjective differential optimization

Jean-Antoine Désidéri*

Project-Team Acumes

Research Report n° 9159 — 21 March 2018 — 41 pages

Abstract: In multiobjective differentiable optimization under constraints, we choose to formulate all type of contraint as an equality constraint, usually nonlinear, possibly by the introduction of a slack variable. Then a predictor-corrector method is proposed. At the predictor, the descent direction is determined by the Multiple-Gradient Descent Algorithm (MGDA) applied to the cost-function gradients projected onto the subspace locally tangent to all constraint surfaces. The step-size is controled to limit the violation of the nonlinear constraints and insure that all cost functions diminish. The corrector permits to restore the nonlinear constraints by a quasi-Newton-type method applied to a function agglomerating all the contraints in which the Hessian is approximated by the sole terms in constraint gradients. This corrector constitutes a quasi-Riemannian approach that reveals very efficient. Thus the predictor-corrector sequence constitutes one iteration of a reduced-gradient descent method for constrained multiobjective optimization. Three classical test-cases are solved for illustration by means of the Inria MGDA software Platform.

Key-words: differentiable multiobjective optimization, nonlinear constraints, restoration, reduced-gradients methods, Newton's method, Riemannian approach

* Directeur de Recherche INRIA, Équipe Acumes

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Algorithme de descente à gradients multiples quasi-riemannien pour l'optimisation différentaible multiobjectif sous contraintes

Résumé : En optimisation différentiable multiobjectif sous contraintes, on fait le choix de formuler tout type de contrainte par une contrainte d'égalité, le plus souvent non linéaire, par le biais d'une variable d'ajustement. On propose alors une méthode prédicteur-correcteur. Au prédicteur, on détermine la direction de descente par l'algorithme de descente à gradients multiples (MGDA) appliqué aux gradients des fonctions coûts projetés dans l'espace localement tangent aux surfaces de contraintes. On contrôle le pas de manière à limiter la violation des contraintes non linéaires et à garantir la diminution de toutes les fonctions coûts. Le correcteur permet de restaurer les contraintes nonlinéaires par une méthode de type quasi-Newton appliquée à une fonction de contrainte agglomérée dans laquelle le hessien est approché par les seuls termes en gradients de contraintes. Ce correcteur constitue une approche quasi riemannienne s'avérant très efficace. Ainsi la séquence prédicteur-correcteur constitue une itération d'une méthode de descente à base de gradients réduits pour l'optimisation multiobjectif sous contraintes. On traite trois cas-tests classiques en illustration au moyen de la plate-forme logicielle MGDA d'Inria.

Mots-clés : optimisation multiobjectif différentiable, contraintes nonlinéaires, restauration, méthode à base de gradients réduits, méthode de Newton, approche riemannienne

1 Introduction

The Multiple-Gradient Descent Algorithm (MGDA) was originally proposed in 2009 for unconstrained, multiobjective differentiable optimization [1] and has been the subject of several publications since. Fundamentally, it relies on the observation that the element of minimum Euclidean norm in the convex hull of the cost-function gradients is unique and constitutes a common ascent direction to all cost functions. The choice of a basis in which the Euclidean norm is defined is free. Thus, by solving the corresponding quadratic programming (QP) problem that defines this vector permits to extend the classical steepest-descent method to the context of multiobjective optimization. When the gradients are linearly independent, the QP problem is solved easily in the canonical basis by a call to a library procedure. Inversely, when the number m of cost functions exceeds, possibly vastly, the dimension n of the admissible set, a hierarchical principle can be introduced to formulate the QP problem in a basis whose convex hull approximates well the convex hull of the complete set of gradients. The QP formulation is then better conditioned and solved without great difficulty. The theory was exposed in [2] and proved efficient in [4] in a problem of active control of a time-periodic flow governed by the compressible Navier-Stokes equations involving 800 gradients of dimension 6. These developments have led to the construction of a software platform open to the research community permitting to operate distantly a procedure that calculates the search direction given the gradients [9].

Although this report focuses on deterministic optimization, note that the simplicity of the MGDA construction has permitted elsewhere to generalize the algorithm to the stochastic framework by considering the stochastic gradient [12] [10].

Returning to the deterministic nonlinear programming context, another line of research has been to investigate Riemannian (or quasi-Riemannian) approches to the treatment of nonlinear equality constraints. A constraint restoration method of quasi-Newton type was proposed in [3] to project the current point onto the constraint manifold by defining an agglomerated constraint function and using an approximate Hessian involving the sole constraint-gradient terms.

In this new report, we show how the MGDA and associated software platform, originally devised for unconstrained problems, can be used as such in conjunction with the quasi-Riemannian approach to solve constrained problems in a predictor-corrector sequence of the family of reduced-gradient-type methods [6]. The constraints are assumed to be of the equality type. However, inequality constraints can be cast as such via slack variables. The present approach is illustrated by the treatment of test-cases of the literature.

2 Problem definition and overview

One considers the following multiobjective minimization problem

$$\min_{\mathbf{x}\in\Omega_a} \{f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})\} \tag{1}$$

subject to the following K equality constraints:

$$\mathbf{c}(\mathbf{x}) = \big(c_1(\mathbf{x}), \ldots, c_K(\mathbf{x})\big) = 0. \tag{2}$$

The functions $\{f_j\}$ $(1 \leq j \leq m)$ and $\{c_k\}$ $(1 \leq k \leq K)$ are defined over the admissible domain $\Omega_a \subseteq \mathbb{R}^n$. They are real-valued, smooth (say $C^2(\Omega_a)$), and admit known gradients, $\{\nabla f_j(\mathbf{x})\}$ and $\{\nabla c_k(\mathbf{x})\}$.

The constraints define a certain manifold $\mathcal{M}$ to which the solution is restricted. Since MGDA is a purely-linear construction, we propose to extend it in the form of a predictor-corrector method in which the predictor accounts for the constraints only linearly, that is, by considering the cost-function gradients projected onto the subspace $\mathcal{T}_\ell$ tangent to the constraint manifold $\mathcal{M}$ at the current iterate, $\mathbf{x}^{(\ell)}$, and the corrector operates orthogonally to it to restore the nonlinear constraints. The following two sections provide the definition of the predictor and corrector steps from the starting point $\mathbf{x}^{(\ell)}$.

3 MGDA predictor step

At $\mathbf{x} = \mathbf{x}^{(\ell)}$, the constraint gradients $\{\nabla c_k(\mathbf{x}^{(\ell)})\}$ $(k = 1, \ldots, K)$ form a family of rank $\mathsf{r} \leq K$. Generally, the constraint gradients are linearly-independent and $\mathsf{r} = K$. In the inverse case, and without great loss of generality, it is assumed that the vectors have been ordered in a way such that $\{\nabla c_k(\mathbf{x}^{(\ell)})\}$ $(k = 1, \ldots, \mathsf{r})$ are linearly-independent, and thus span $\mathcal{N}_\ell = \mathcal{T}_\ell^\perp$.

The QR factorization (see e.g. [14]) is applied to the $n \times \mathsf{r}$ matrix whose column-vectors are the (linearly-independent) constraint gradients:

$$\mathbf{J}_c = \nabla \mathbf{c}(\mathbf{x}^{(\ell)}) = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \nabla c_1(\mathbf{x}^{(\ell)}) & \nabla c_2(\mathbf{x}^{(\ell)}) & \ldots & \nabla c_{\mathsf{r}}(\mathbf{x}^{(\ell)}) \\ \vdots & \vdots & & \vdots \end{pmatrix} = \mathbf{Q}\,\mathbf{R}. \tag{3}$$

The column-vectors of matrix $\mathbf{Q}$ are denoted $\{\mathbf{q}_k\}$ $(k = 1, \ldots, \mathsf{r})$:

$$\mathbf{Q} = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \mathbf{q}_1 & \mathbf{q}_2 & \ldots & \mathbf{q}_{\mathsf{r}} \\ \vdots & \vdots & & \vdots \end{pmatrix} \tag{4}$$

They form an orthonormal basis of the span of the contraint gradients. The matrix $\mathbf{R}$ is $\mathsf{r} \times \mathsf{r}$, upper triangular and invertible. The matrices $\mathbf{Q}$ and $\mathbf{R}$ can be determined numerically by using, for example, the Lapack procedure DGEQRF of the Netlib Library.

The cost-function gradients are calculated:

$$\mathbf{g}_j = \nabla f_j \left(\mathbf{x}^{(\ell)}\right). \tag{5}$$

Optionally, these vectors can be scaled component-wise and/or by the function values $\{f_j(\mathbf{x}^{(\ell)})\}$, if logarithmic gradients are preferred (see the recommendations in [9]). To simplify the description of the present method, these options are not included in the present description, although not excluded, and possibly useful, in the application of the MGDA process.

Then, these gradients are projected onto the subspace $\mathcal{T}_\ell$ to get:

$$\mathbf{g}'_j = \mathbf{g}_j - \sum_{k=1}^{\mathsf{r}} \left(\mathbf{q}_k^t\, \mathbf{g}_j\right) \mathbf{q}_k \qquad (j = 1, \ldots, m). \tag{6}$$

The MGDA procedure is applied to these projected gradients to calculate the direction of search $\mathbf{d}_\ell^\star \in \mathcal{T}_\ell$, and in order to complete the definition of predicted vector $\mathbf{x}$ denoted $\bar{\mathbf{x}} = \mathbf{x}^{\overline{\ell+1}}$, an appropriate step-size $\epsilon_\ell^\star$ remains to be defined :

$$\bar{\mathbf{x}} = \mathbf{x}^{\overline{\ell+1}} = \mathbf{x}^{(\ell)} - \epsilon_\ell^\star \mathbf{d}_\ell^\star. \tag{7}$$

This is accomplished in several steps. A first provisional estimate for vector $\bar{\mathbf{x}}$ is calculated using the user-supplied reference step-size ϵ_0:

$$\bar{\mathbf{x}}_0 = \mathbf{x}^{(\ell)} - \epsilon_0 \mathbf{d}_\ell^\star. \tag{8}$$

The reference step-size ϵ_0 should be of the expected order of magnitude of $\epsilon_\ell^\star$, preferably by excess. The constraints are then evaluated at this point, as well as the maximum:

$$c_{\max} = \max_k \left| c_k\left(\bar{\mathbf{x}}_0\right)\right|. \tag{9}$$

Since the vector $\mathbf{d}_\ell^\star$ is orthogonal to all constraint gradients, the following approximation holds (for all k):

$$c_k(\mathbf{x}^{(\ell)} - \epsilon \mathbf{d}_\ell^\star) = c_k\left(\bar{\mathbf{x}}_0\right) \left(\frac{\epsilon}{\epsilon_0}\right)^2 + O(\epsilon^3) \tag{10}$$

for small ϵ. Note that an excessive violation of the constraints by the MGDA predictor step could cause an undue difficulty in the subsequent corrector-step constraint-restoration procedure. To avoid this, given a user-supplied bound B, a new step-size ϵ_1 is defined as follows:

$$\epsilon_1 = \begin{cases} \epsilon_0 \text{ if } c_{\max} \leq B \\ \epsilon_0 \sqrt{\frac{B}{c_{\max}}} \text{ otherwise} \end{cases} \tag{11}$$

and the following new provisional estimate is calculated:

$$\bar{\mathbf{x}}_1 = \mathbf{x}^{(\ell)} - \epsilon_1 \mathbf{d}_\ell^\star. \tag{12}$$

Then, the actual step-size is chosen to insure that all cost-functions diminish. For this, for each cost-function $f_j(\mathbf{x})$ $(j = 1, \ldots, m)$, one computes $f_{j,1} = f_j(\bar{\mathbf{x}}_1)$ and compares it with the linear extrapolate $\tilde{f}_{j,1} = f_{j,0} - \epsilon_1 \sigma_j$ where $f_{j,0} = f_j\left(\mathbf{x}^{(\ell)}\right)$ and σ_j is the directional derivative $\sigma_j = \left(\mathbf{g}_j'\right)^t \mathbf{d}_\ell^\star$. By virtue of the MGDA construction, $\sigma_j > 0$ and it is often true that σ_j is independent of j. If $f_j(\mathbf{x})$ is locally strictly-convex, $\tilde{f}_{j,1} < f_{j,1}$, and the second-degree function

$$\varphi_j(\epsilon) = f_{j,0} - \sigma_j \epsilon + (f_{j,1} - f_{j,0} + \sigma_j \epsilon_1) \left(\frac{\epsilon}{\epsilon_1}\right)^2 \tag{13}$$

is such that

$$\varphi_j(0) = f_{j,0}, \quad \varphi_j'(0) = -\sigma_j, \quad \varphi_j(\epsilon_1) = f_{j,1}, \tag{14}$$

and it achieves a minimum at

$$\epsilon = \bar{\epsilon}_j = \frac{\sigma_j \epsilon_1^2}{2\left(f_{j,1} - f_{j,0} + \sigma_j \epsilon_1\right)}. \tag{15}$$

If inversely, $f_{j,1} \leq \bar{f}_{j,1}$, it is assumed that the function $f_j(\mathbf{x})$ is not locally strictly-convex. Then $f_{j,1} < f_{j,0}$, and one defines conservatively $\bar{\epsilon}_j = \epsilon_1$.

Then one lets provisionally:

$$\epsilon_\ell^\star = \min\left\{\epsilon_1, \bar{\epsilon}_1, \ldots, \bar{\epsilon}_m\right\}. \tag{16}$$

Lastly, if $\epsilon_\ell^\star = \epsilon_0$, the parameter ϵ_0, is judged *a posteriori* too small. Then, ϵ_0 is doubled and the whole process of step-size selection is conducted again, starting from (11). Otherwise, the above estimate $\epsilon_\ell^\star$ is accepted, and this completes the definition of the predictor step.

Thereafter, to simplify the notation, one lets $\bar{\mathbf{x}} = \mathbf{x}^{\overline{\ell+1}}$. In general, this point does not satisfy the nonlinear constraints (to a specified tolerance). Hence a corrector step is to be performed to restart from a point strictly on the manifold $\mathcal{M}$.

4 Quasi-Newton Riemannian-restoration corrector step

While the predictor step has been performed in the subspace $\mathcal{T}_\ell$ locally tangent to the constraint manifold $\mathcal{M}$, the corrector step results from one or more iterations, each iteration being performed in the span of the (adaptive) constraint gradients, or equivalently, the orthonormal vectors $\{\mathbf{q}_k\}$.

The method is one of "restoration" type [6] (Chapter 6). Following the report [3], a single agglomerated constraint function is defined:

$$\gamma(\mathbf{x}) = \tfrac{1}{2}\mathbf{c}(\mathbf{x})^t \mathbf{c}(\mathbf{x}) = \sum_{k=1}^{\mathsf{r}} \tfrac{1}{2} c_k(\mathbf{x})^2. \tag{17}$$

In the above writing, it is assumed that prior to this definition, the different constraint functions $\{c_k(\mathbf{x})\}$ have been scaled appropriately, perhaps adaptively, that is, depending on the contraint values $\{c_k(\bar{\mathbf{x}})\}$ that are not all equal to 0 or below a tolerance, and possibly other local information.

It is proposed to minimize γ (up to the limit $\gamma \to 0$) by a pseudo-Newton iteration. For this purpose, the formal expression of the gradient of $\gamma(x)$ is calculated

$$\nabla\gamma = \sum_{k=1}^{\mathsf{r}} c_k(\mathbf{x})\nabla c_k(\mathbf{x}) \tag{18}$$

as well as the corresponding Hessian matrix:

$$H = \nabla^2\gamma = \mathcal{H} + \sum_{k=1}^{\mathsf{r}} c_k(\mathbf{x})\nabla^2 c_k(\mathbf{x}) \tag{19}$$

where

$$\mathcal{H} = \sum_{k=1}^{\mathsf{r}} \left[\nabla c_k(\mathbf{x})\right]\left[\nabla c_k(\mathbf{x})\right]^t. \tag{20}$$

At $\mathbf{x} = \mathbf{x}^{(\ell)}$, the constraints are satisfied, and $\nabla\gamma = 0$ and $H = \mathcal{H}$. Hence, at a neighboring point, in the above expression of the Hessian, (19), the second term which involves the Hessians of the constraint functions weighted by the constraint values themselves, is much smaller than the first, $\mathcal{H}$: in the limit $\epsilon_\ell \to 0$, the first term is $O(1)$, while the second is $O(\epsilon_\ell^2)$. Hence

$$H \doteq \mathcal{H}. \tag{21}$$

In order to apply a pseudo-Newton-type method to solve the equation $\mathbf{c}(\mathbf{x}) = 0$ by minimizing $\gamma(x)$ from the starting point $\bar{x}$, one considers the iteration

$$\begin{cases} \mathbf{y}^{(0)} = \bar{\mathbf{x}} \\ \mathbf{y}^{(\lambda+1)} = \mathbf{y}^{(\lambda)} + \delta\mathbf{y}^{(\lambda)} \end{cases} \tag{22}$$

$(\lambda = 0, 1, \dots)$, where

$$\mathcal{H}^{(\lambda)}\delta\mathbf{y}^{(\lambda)} = -\nabla\gamma^{(\lambda)}, \tag{23}$$

in which $\mathcal{H}^{(\lambda)} = \mathcal{H}\left(\mathbf{y}^{(\lambda)}\right)$ is the defined-above approximate Hessian matrix, and $\nabla\gamma^{(\lambda)} = \nabla\gamma\left(\mathbf{y}^{(\lambda)}\right)$.

However (23) requires some precision to be made since matrix $\mathcal{H}$ is not full rank. Despite this deficiency, (23) admits a unique solution in the subspace spanned by the constraint gradients [3]. Hence we propose to define the corrector step as the fixed point of the iteration defined in (22)-(23) when using this unique solution. For sake of completeness, we provide here the detailed formulas used in the solution of (23).

At each pseudo-Newton's iteration, the QR factorization of the local constraint gradients is performed to get updated vectors $\{\mathbf{q}_k\}$. Omitting now the superscript $^{(\ell)}$ to alleviate the notation, one seeks for $\delta\mathbf{y} = \delta\mathbf{y}^{(\ell)}$ in the span of these vectors

$$\delta\mathbf{y} = \sum_{k=1}^{K} \eta_k\mathbf{q}_k = \mathbf{Q}\eta \tag{24}$$

where the vector $\{\eta\}$ is now the new unknown. Note that

$$\nabla c_i = \mathbf{J}_c\mathbf{e}_i = \mathbf{Q}\mathbf{R}\mathbf{e}_i \qquad (i = 1, \dots, \mathsf{r}) \tag{25}$$

where $\mathbf{e}_i$ is the ith element of the canonical basis of $\mathbb{R}^{\mathsf{r}}$. Hence

$$\mathcal{H}\,\delta\mathbf{y} = \left(\sum_{i=1}^{\mathsf{r}} \left[\nabla c_i\right]\left[\nabla c_i\right]^t\right)\mathbf{Q}\eta = \left(\sum_{i=1}^{\mathsf{r}} \left[\nabla c_i\right]\mathbf{e}_i^t\mathbf{R}^t\mathbf{Q}^t\right)\mathbf{Q}\eta = \sum_{i=1}^{\mathsf{r}} \left(\mathbf{e}_i^t\mathbf{R}^t\eta\right)\nabla c_i. \tag{26}$$

But

$$\nabla\gamma = \sum_{i=1}^{\mathsf{r}} c_i\nabla c_i \tag{27}$$

and since the vectors $\{\nabla \mathbf{c}_i\}$ $(i = 1, \ldots, \mathsf{r})$ are linearly-independent, (23) is equivalent to the following system of linear equations for the unknown vector η:

$$\mathbf{e}_i^t \mathbf{R}^t \eta = -c_i \qquad (\forall i), \tag{28}$$

that is:

$$\mathbf{R}^t \eta = -\mathbf{c} \tag{29}$$

whose inversion completes the definition of the incremental vector $\delta\mathbf{y}$. Since matrix $\mathbf{R}$ is upper triangular and invertible, the inversion is done readily by forward substitution.

The pseudo-Newton iteration is continued until the constraints are satisfied sufficiently accurately. A convergence criterion can be of the following type

$$c_{\max} \leq TOL \tag{30}$$

where TOL is specified tolerance. If this is achieved by $\mathbf{y}^{(\lambda)}$, one sets

$$\mathbf{x}^{(\ell+1)} = \mathbf{y}^{(\lambda)} \tag{31}$$

which completes the predictor-corrector sequence.

Remark 1
The user-supplied bound B in the predictor step and the user-supplied tolerance parameter TOL in the corrector step can be defined in relation to one another. If TOL is very small, a rule of thumb may be to let $B = 100 \times TOL$, or more.

5 Summary of the numerical method

MGDA Predictor step from $\mathbf{x}^{(\ell)}$ to $\bar{\mathbf{x}} = \mathbf{x}^{\overline{\ell+1}}$ (outer loop)

- Compute constraint gradients, perform QR factorization, (3), and identify orthogonal basis $\{\mathbf{q}_j\}$ $(j = 1, \ldots, \mathsf{r})$ of their span.
- Compute cost-function gradients, (5), and their projections onto the subspace locally tangent to the constraint manifold, (6).
- Compute ascent vector $\mathbf{d}_\ell^\star$ by a call to *mgdaProcess* using the projected gradients as inputs.
- Select step-size $\epsilon_\ell^\star$; for this:
  - Compute maximum constraint violation at $\bar{\mathbf{x}}_0$ given by (8) in which ϵ_0 is the user-supplied reference step-size, to be provided by excess.
  - Compute limited estimate ϵ_1 by (11) in which B is a user-supplied bound on the constraint violation.
  - Compute cost function values at $\bar{\mathbf{x}}_1$ given by (12); for each function, estimate associated optimal step-size $\bar{\epsilon}_j$.
  - Set $\epsilon_\ell^\star$ using (16). If $\epsilon_\ell^\star = \epsilon_0$, double ϵ_0 and restart the whole step-size selection; otherwise proceed with the definite predictor update.
- Update optimization vector according to (7).

Restoration corrector by quasi-Newton inner iteration from $\bar{\mathbf{x}}$ to $\mathbf{x}^{(\ell+1)}$ (inner loop) Perform inner iteration initiated at the defined-above predicted vector $\mathbf{y}^{(0)} = \bar{\mathbf{x}}$, in which $\mathbf{y}^{(\lambda+1)}$ ($\lambda = 0, 1, \dots$) is computed as follows:

- Compute constraint values $\{c_k\}$ (stored in vector $\mathbf{c}$), and constraint gradients, and perform QR factorization.
- Interrupt iteration if $|c_k| \leq TOL$ $(\forall k)$ where TOL is the user-supplied constraint violation tolerance. Then $\mathbf{x}^{(\ell+1)} = \mathbf{y}^{(\lambda)}$; otherwise:
- Solve triangular system (29) for vector η by forward substitution.
- Synthetize incremental vector $\delta\mathbf{y}$ by (24) and update vector $\mathbf{y}$ according to (22), and start a new inner iteration.

6 Examples

6.1 Test Problem 47 from the Hock-Schittkowski collection

In the Hock-Schittkowski collection [7], Test Problem 47 refers to [8] and [11]. The test-case consists of the minimization of a quartic function of 5 variables subject to 3 equality constraints, one of which is cubic and the other two quadratic:

$$\min f(\mathbf{x}) = (x_1 - x_2)^2 + (x_2 - x_3)^3 + (x_3 - x_4)^4 + (x_4 - x_5)^4 \tag{32}$$

subject to:

$$\begin{cases} x_1 + x_2^2 + x_3^3 - 3 = 0 \\ x_2 - x_3^2 + x_4 - 1 = 0 \\ x_1 x_5 - 1 = 0. \end{cases} \tag{33}$$

The starting point is feasible and specified

$$\mathbf{x}^{(0)} = (2, \sqrt{2}, -1, 2 - \sqrt{2}, \tfrac{1}{2}) \tag{34}$$

so that $f(\mathbf{x}^{(0)}) \doteq 20.7381$ (given erroneously in [7]), and the minimizing point is

$$\mathbf{x}^\star = (1, 1, 1, 1, 1) \tag{35}$$

for which $f(\mathbf{x}^\star) = 0$.

This problem is single-objective; thus here MGDA reduces to the classical steepest-descent method, and the MGDA Platform is not used. The constraints are nonlinear and all three of equality type. Hence the problem is exactly cast in the format of the present report. By treating this test-case, we wish to demonstrate the viability of the proposed restoration procedure.

The methodological parameters were assigned the following values:

– Reference step-size, $\epsilon_0 = 0.1$.

– Constraint-violation bound at the predictor step, $B = 1$.

– Constraint-violation tolerance at the corrector step, $TOL = 10^{-4}$.

– Maximum allowable number of quasi-Newton (inner-loop) iterations: 20.

As a result, the cost-function converged to visual accuracy in 6 predictor-corrector iterations as illustrated by Figure 1. The number of inner-loop quasi-Newton's iterations was found to be successively: 10, 2, 6, 4, 3, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2. Thus, the projection onto the constraint manifold was found difficult over the first four iterations, but very easy subsequently.

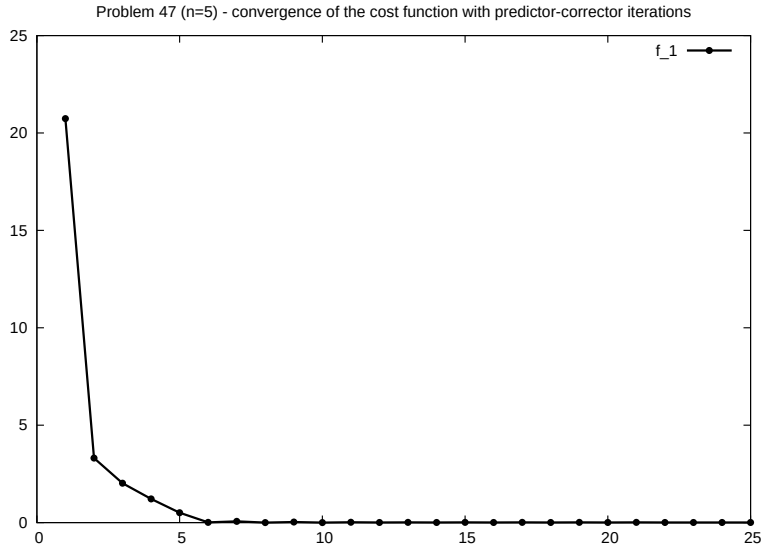


Figure 1: Test Problem 47: convergence of the cost-function

Figure 2 indicates the convergence of the cost function and the norm of the ascent step, that is here, for the steepest-descent method, the norm of the gradient, in a log scale. The cost function is reduced by 3 orders of magnitude from the initial value, the gradient less, unsurprisingly. A small oscillation is eventually triggered as confirmed by the convergence of the optimization variables (Figure 3).

To get rid of the parasitic tail-convergence oscillation, a second experiment was conducted using the same setting of parameters, but by including under-relaxation over the optimization outer loop:

$$\mathbf{x}^{(\ell+1)} = \omega_r \widetilde{\mathbf{x}^{\ell+1}} + (1-\omega_r)\mathbf{x}^{(\ell)} \tag{36}$$

where $\mathbf{x}^{(\ell)}$ is the ℓth iterate, $\widetilde{\mathbf{x}^{\ell+1}}$ the next iterate by the standard predictor-corrector method (without under-relaxation), and $\mathbf{x}^{(\ell+1)}$ the definite $\ell+1$st iterate by the (under-relaxed) method. The relaxation parameter ω_r was progressively reduced with iterations; at iteration ℓ: $\omega_r = (\ell_{\max} - \ell + 1)/\ell_{\max}$ (in the experiment $\ell_{\max} = 25$). Figures 4, 5 and 6 are analogous to Figures 1, 2 and 3 and correspond to the experiment including under-relaxation.

The under-relaxation eliminates very well the problem with the tail-convergence oscillation. In 25 iterations, the cost function is reduced by more than 6 orders of magnitude (from 20.7381 to 7×10^{-6}), and the tail convergence is smooth in both function values and optimization variables. Lastly, the number of quasi-Newton iterations was found to be successively 10, 2, 6, 5, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0.

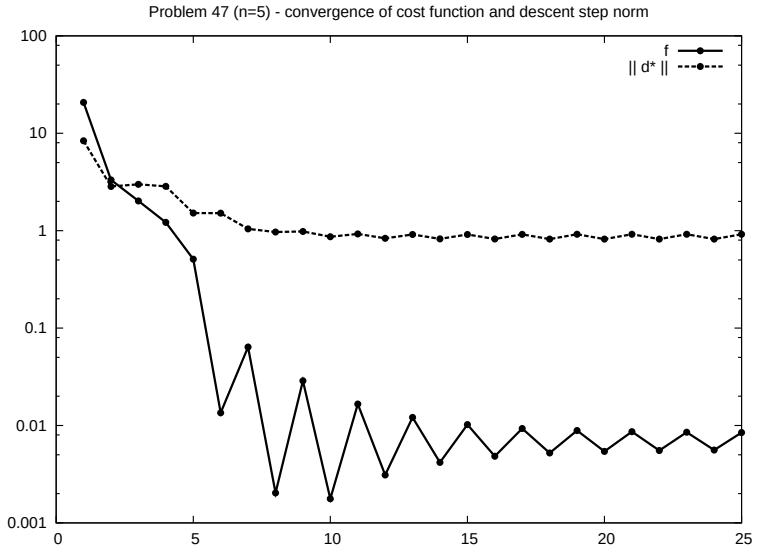


Figure 2: Test Problem 47: convergence of the cost-function

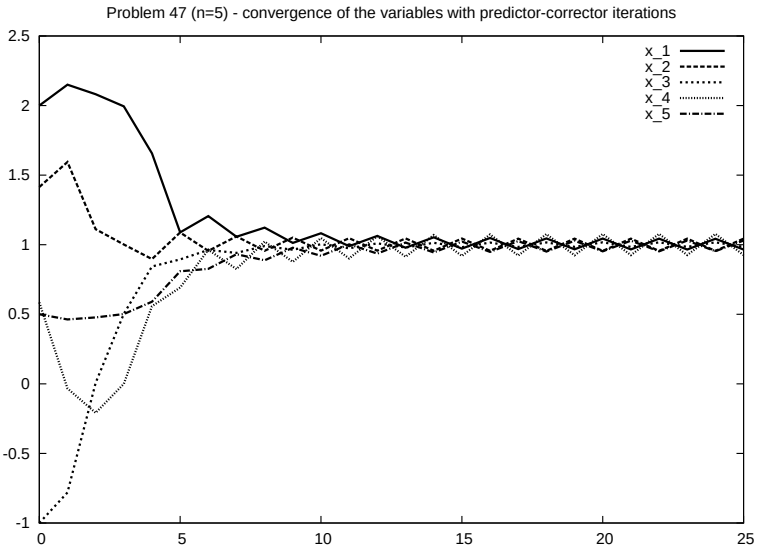


Figure 3: Test Problem 47: convergence of the optimization variables

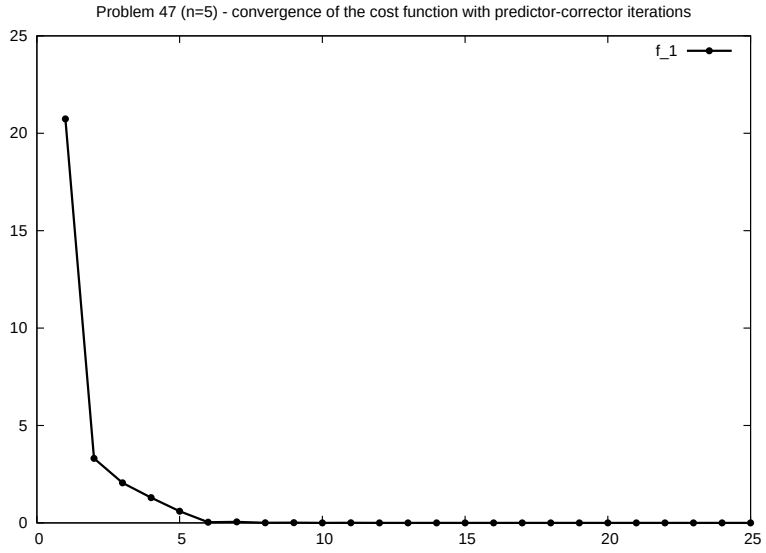


Figure 4: Test Problem 47: convergence of the cost-function with under-relaxation

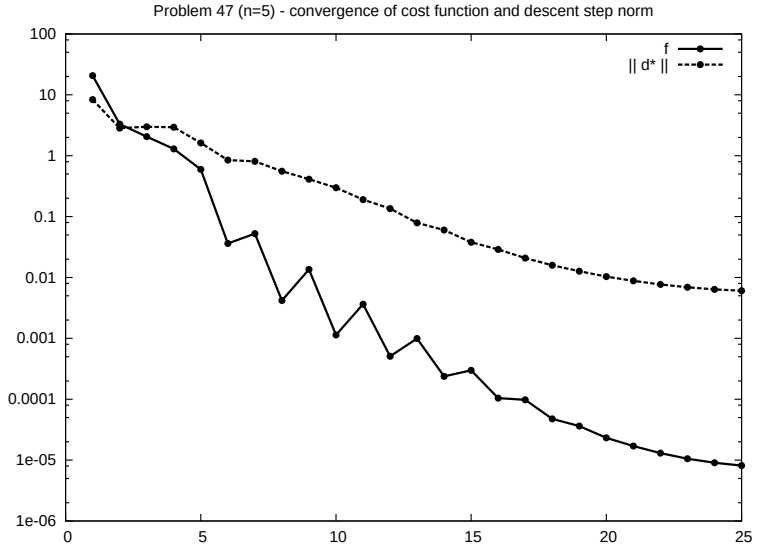


Figure 5: Test Problem 47: convergence of the cost-function with under-relaxation

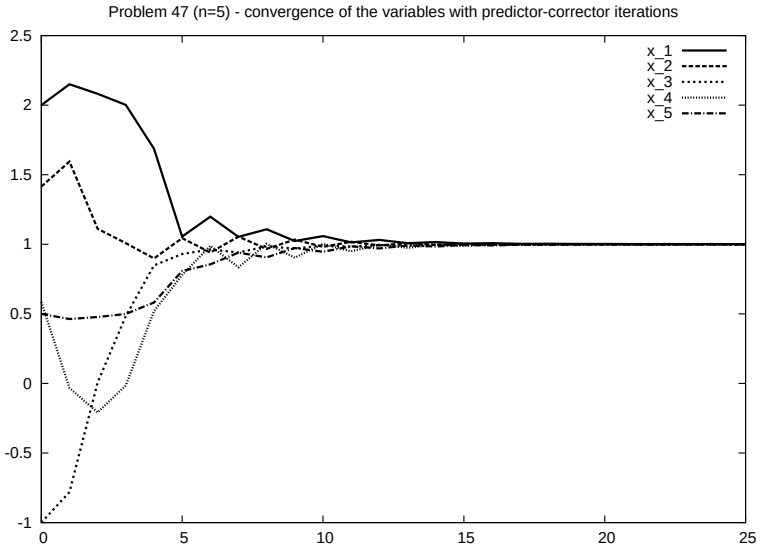


Figure 6: Test Problem 47: convergence of the optimization variables with under-relaxation

6.2 Rosenbrock test-case subject to constraints

One considers the minimization of the Rosenbrock function constrained with a cubic and a line [13]:

$$\min f_1(\mathbf{x}) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \tag{37}$$

subject to the following bounds on the two variables:

$$-1.5 \leq x_1 \leq 1.5 \tag{38}$$

$$-0.5 \leq x_2 \leq 2.5 \tag{39}$$

and the following two inequality constraints:

$$(x_1 - 1)^3 - x_2 + 1 < 0 \tag{40}$$

$$x_1 + x_2 - 2 < 0. \tag{41}$$

This optimization test-case is single-objective. It is considered here to illustrate the transposition of the inequality constraints into equality constraints via the introduction of slack variables, and the convergence of the restoration procedure in a case fabricated to involve partial derivatives of very different magnitudes ("deep valley").

The iso-value contours of the Rosenbrock function are depicted in blue on Figure 7, and the segment and the curve arc associated with the linear constraint and the cubic constraint respectively are indicated in red. The admissible domain is on the left of the boundary. On Figure 8, the variations of the cost function with x_2 along the boundary are displayed. The function admits a local minimum of 1 at $x_1 = x_2 = 0$, and a global minimum of 0 at $x_1 = x_2 = 1$. In between these two points the function admits a maximum, which is the stem of a ridge line in the two-dimensional domain. Hence, depending on the position of the starting point with respect to the ridge line, the steepest-descent method is expected to converge to the local or the global minimum.

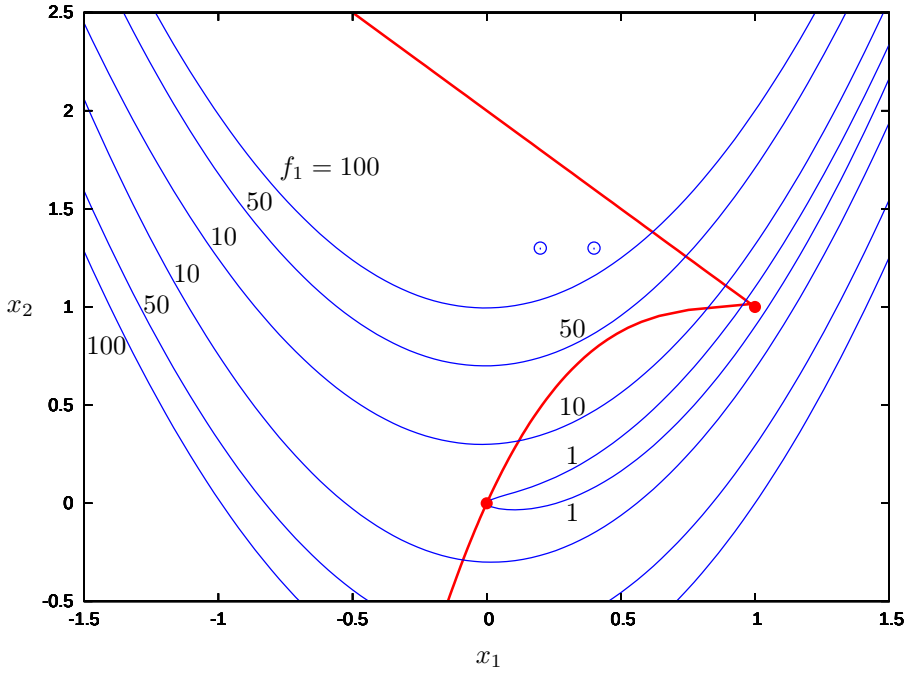


Figure 7: Iso-value contours of the Rosenbrock function constrained with a cubic and a line. The admissible domain is situated to the left of the constraint boundary (in red).

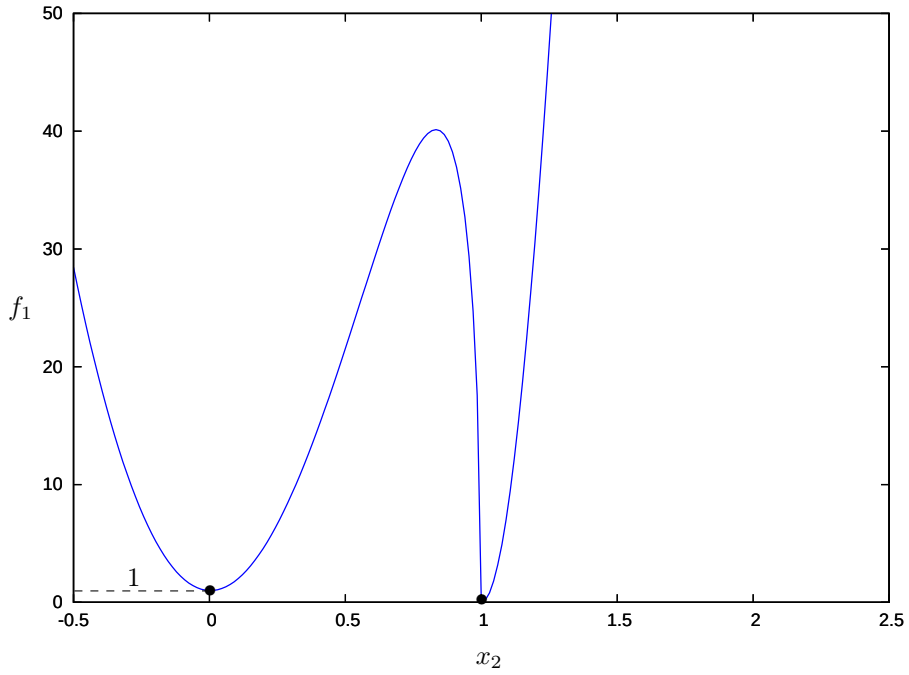


Figure 8: Variations with x_2 of the Rosenbrock function along the constraint boundary.

Evidently, both minimums violate (40). Therefore, strictly speaking, the problem as stated in [15] is ill-posed. To remedy this, the inequality constraints are now considered in the large sense:

$$(x_1-1)^3 - x_2 + 1 \leq 0 \tag{42}$$

$$x_1 + x_2 - 2 \leq 0. \tag{43}$$

Then many possibilities exist to satisfy all four constraints by the introduction of four slack variables (x_3, x_4, x_5, x_6). For example, recalling that the hyperbolic cosine function maps the real line into $[1, +\infty]$, one can let:

$$x_1 = 1.5 \sin x_3 \tag{44}$$

$$x_2 = 1 + 1.5 \sin x_4 \tag{45}$$

$$(x_1-1)^3 - x_2 + 1 = 1 - \cosh x_5 \tag{46}$$

$$x_1 + x_2 - 2 = 1 - \cosh x_6. \tag{47}$$

In other words, the minimization of the function $f_1(\mathbf{x})$ subject to the above four inequality constraints is going to be conducted in $\mathbb{R}^6$ and subject to the following four nonlinear equality contraints:

$$c_1(\mathbf{x}) = x_1 - 1.5 \sin x_3 = 0 \tag{48}$$

$$c_2(\mathbf{x}) = x_2 - 1 - 1.5 \sin x_4 = 0 \tag{49}$$

$$c_3(\mathbf{x}) = (x_1-1)^3 - x_2 + \cosh x_5 = 0 \tag{50}$$

$$c_4(\mathbf{x}) = x_1 + x_2 - 3 + \cosh x_6 = 0. \tag{51}$$

A feasible starting point (x_1, x_2) is provided by the user, satisfying all four inequality constraints. Then the four slack variables are computed from:

$$x_3 = \sin^{-1}(\frac{x_1}{1.5}) \tag{52}$$

$$x_4 = \sin^{-1}(\frac{x_2 - 1}{1.5}) \tag{53}$$

$$x_5 = \cosh^{-1}\big(-(x_1-1)^3 + x_2\big) \tag{54}$$

$$x_6 = \cosh^{-1}\big(3 - x_1 - x_2\big) \tag{55}$$

where for $y \geq 1$, $\cosh^{-1} y = \ln\Big(y + \sqrt{y^2-1}\Big)$.

Here, for a single-objective optimization, MGDA reduces to the steepest-descent method. It is well-known that this method is not robust enough to converge to the global minimum unless the starting point is on the right side of the pass. Nevertheless, the steepest-descent method is a valid base formulation for a broad class of problems, and by treating this test-case, it is not our purpose to address the specific issue of robustness inherent to the steepest-descent method, but only to show that the present numerical method performs correctly in handling various nonlinearities, in particular in the proposed quasi-Newton constraint restoration procedure.

Indeed, a successful convergence was obtained from the starting point

$$(x_1, x_2) = (0.4, 1.3) \tag{56}$$

indicated by a open symbol on Figure 7, by using the following setting of parameters:

– Reference step-size, $\epsilon_0 = 0.05$.

– Constraint-violation bound at the predictor step, $B = 0.1$.

– Constraint-violation tolerance at the corrector step, $TOL = 10^{-4}$.

– Maximum allowable number of quasi-Newton (inner-loop) iterations: 4.

The convergence of the 6 optimization variables is indicated on Figure 9. Visual convergence of all variables is achieved in say 6 iterations. Both variables x_1 and x_2 converge to 1, for which the global minimum is reached. The slack variable x_3 converges to $\sin^{-1}(2/3) \doteq 0.73$, and the other 3 slack variables to 0. Somewhat surprisingly, the slack variable x_6 associated with the linear constraint converges slightly less rapidly. Figure 10 depicts the iterative convergence of the cost-function f_1 and of the norm of the ascent vector $\|\mathbf{d}_\ell^\star\|$. In 11 iterations, $f_1 \doteq 10^{-20}$. As mentioned before, $\mathbf{d}_\ell^\star = \nabla f_1$ and the norm converges more slowly than f_1, unsurprisingly.

Concerning the quasi-Newton (inner-loop) iteration, with the adopted setting of the parameters, it is interesting to observe how the step-size has been controled. Over the first 4 iterations the constraint-violation limitation (11) alone was active. At iteration 5, both limitations (11) and (16) were active, the second after the first. At iteration 6, the constraint-violation limitation (11) was not active, and never active again thereafter, and the step-size was given by (16). At iterations 8, 9 and 10, neither limitation was active with the initial ϵ_0. Hence ϵ_0 was doubled, in fact twice at all three iterations, and finally reached the value 12.8. Then, from iteration 11 on, the step-sizes ϵ_0 and $\epsilon_\ell^\star$ remained constant. The number of quasi-Newton iterations was found successively equal to 1, 1, 1, 2, 1, 1, 1, 1 and 0 afterwards. Thus from iteration 9 on, global convergence had achieved a sufficient level for the constraint enforcement to be abandoned. In conclusion, the inner-loop convergence is very fast, and the restoration technique is validated at least in situations favorable to the steepest-descent method.

We now consider numerical experiments using the following new starting point

$$(x_1, x_2) = (0.2, 1.3) \tag{57}$$

also indicated by an open symbol on Figure 7. This new starting point is close to the previous one, but it is situated on the unfavorable side of the previously-mentioned ridge. Consequently, convergence is again achieved but to the local minimum corresponding to $x_1 = x_2 = 0$ (instead of 1) and $f_1 = 1$ (instead of 0).

With the same setting of parameters, in 25 iterations f_1 achieves the value: 1.0005764516600555. The corresponding convergence plot for the variables is provided by Figure 11, and for the pair $(f_1, \|\mathbf{d}_\ell^\star\|)$ by Figure 12. A small oscillation is observed in $\|\mathbf{d}_\ell^\star\|$. Nevertheless the function f_1 has been reduced of some 6 orders of magnitude in 25 steps.

For the same case, a superior iterative convergence performance can be achieved with a slightly different setting of parameters. The constant B fixing the constraint-violation tolerance at the predictor step was reduced to $B = 10^{-2}$, 100 iterations were performed and under-relaxation as defined by (36) was applied with $\omega_r = \sqrt{(\ell - 101)/100}$ (ℓ: iteration count). The corresponding convergence plots are given in Figure 13 for the variables and in Figure 14 for the cost function. One observes a better convergence of the slack variable x_5 that controls the satisfaction of the nonlinear constraint. In 100 iterations, the cost function achieved, by successive steps observed in the convergence, a slightly different value $f_1^\star \doteq 0.99885759495058235$. In total, the difference $|f_1 - f_1^\star|$ was iteratively reduced here of nearly 13 orders of magnitude.

In summary, we conclude to the viability of the formulation of constraints via slack variables, and efficiency of the quasi-Newton restoration technique. Global convergence is achieved if the starting point is appropriate.

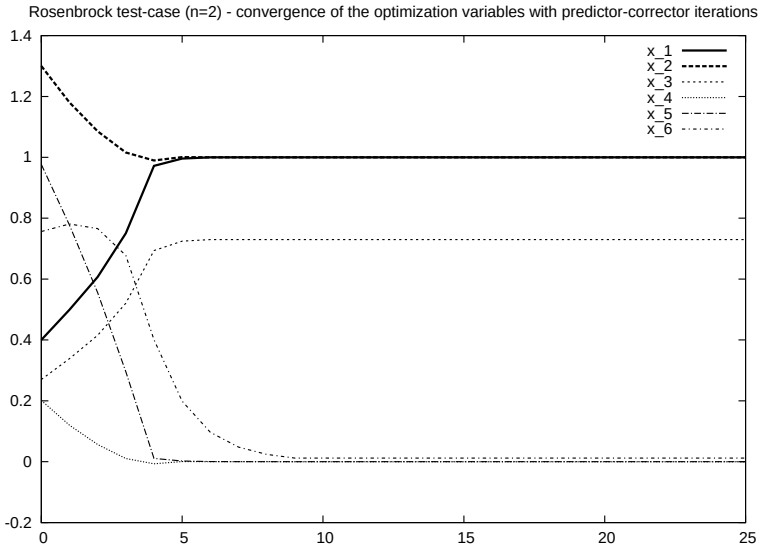


Figure 9: Rosenbrock test-case: convergence of the optimization variables with predictor-corrector iterations. (Starting point: $(x_1, x_2) = (0.4, 1.3)$.)

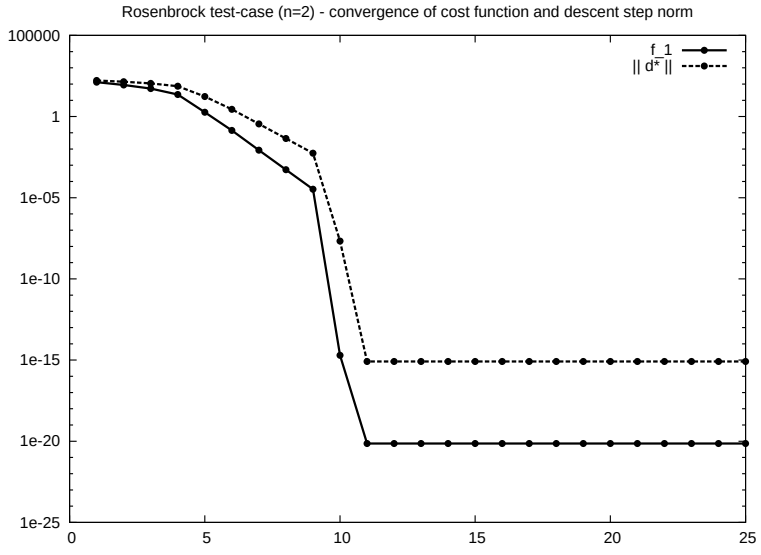


Figure 10: Rosenbrock test-case: convergence of the cost function f_1 and norm of the ascent vector $\|\mathbf{d}_\ell^\star\|$ with predictor-corrector iterations. (Starting point: $(x_1, x_2) = (0.4, 1.3)$.)

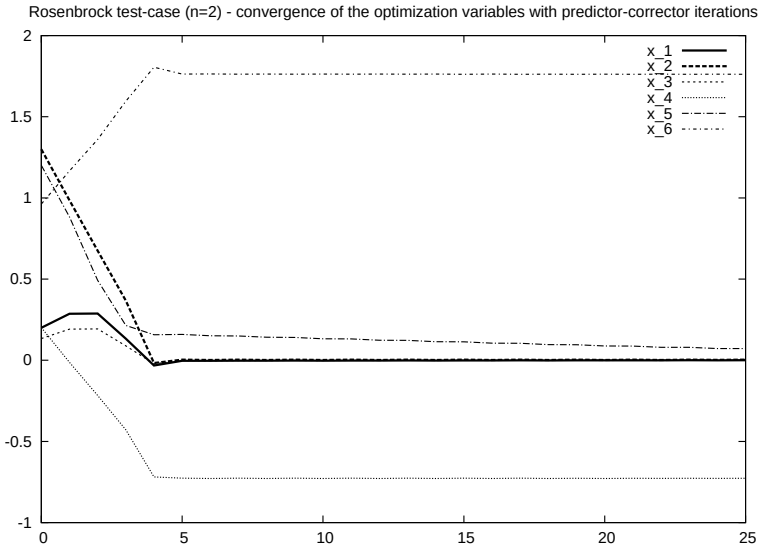


Figure 11: Rosenbrock test-case: convergence of the optimization variables with predictor-corrector iterations. (Starting point: $(x_1, x_2) = (0.2, 1.3)$.)

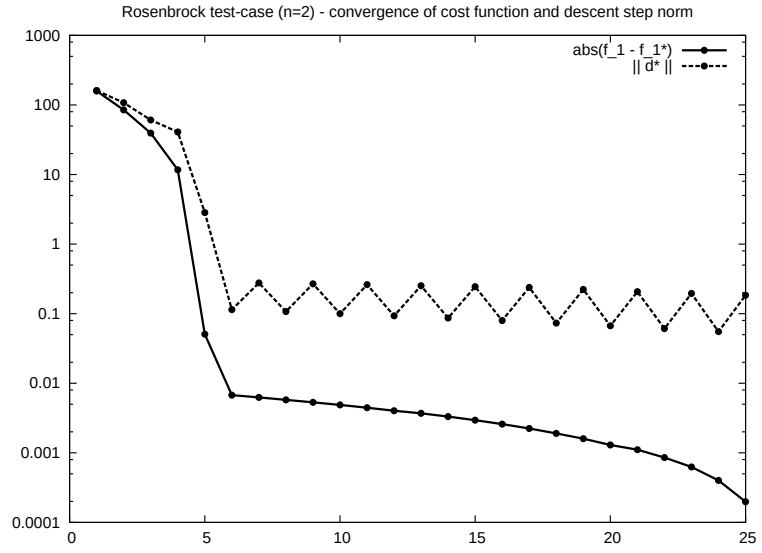


Figure 12: Rosenbrock test-case: convergence of the cost function f_1 and norm of step-size $\|\mathbf{d}_\ell^\star\|$ with predictor-corrector iterations. (Starting point: $(x_1, x_2) = (0.2, 1.3)$.)

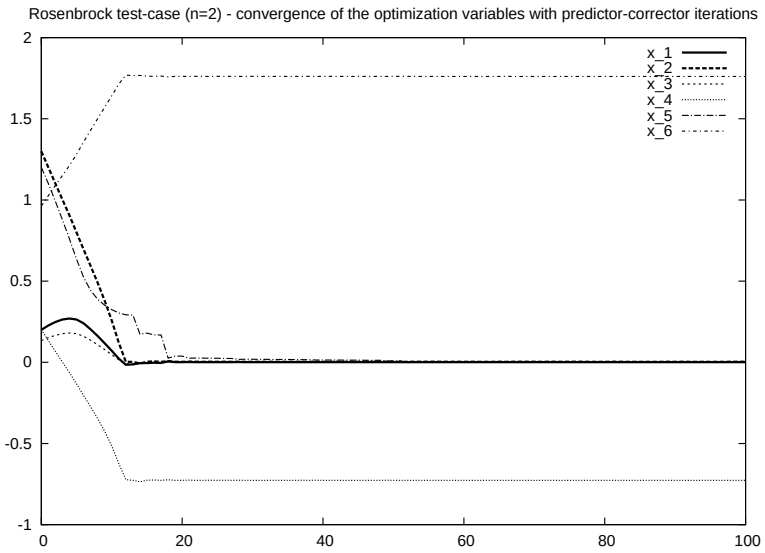


Figure 13: Rosenbrock test-case: convergence of the optimization variables with predictor-corrector iterations. (Starting point: $(x_1, x_2) = (0.2, 1.3)$; reduced $B = 0.01$ and under-relaxation.)

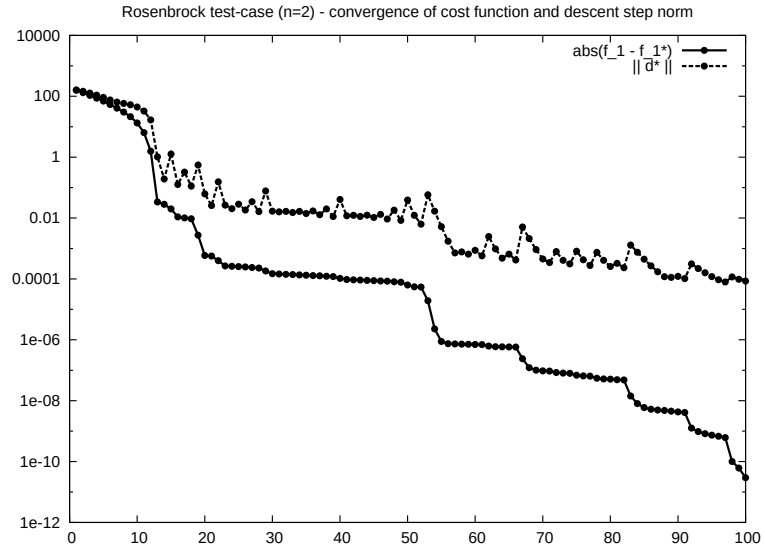


Figure 14: Rosenbrock test-case: convergence of the cost function f_1 and norm of step-size $\|\mathbf{d}_\ell^\star\|$ with predictor-corrector iterations. (Starting point: $(x_1, x_2) = (0.2, 1.3)$; reduced $B = 0.01$ and under-relaxation.)

6.3 Fonseca-Fleming test-case

One considers now the Fonseca-Fleming test-case [5] in the case of $n = 2$ variables. The problem is the minimization of the following two cost-functions:

$$f_1(\mathbf{x}) = 1 - \exp\left[-\left(x_1 - \frac{1}{\sqrt{2}}\right)^2 - \left(x_2 - \frac{1}{\sqrt{2}}\right)^2\right] \tag{58}$$

$$f_2(\mathbf{x}) = 1 - \exp\left[-\left(x_1 + \frac{1}{\sqrt{2}}\right)^2 - \left(x_2 + \frac{1}{\sqrt{2}}\right)^2\right] \tag{59}$$

subject to the bound constraints

$$-4 \leq x_1, x_2 \leq 4. \tag{60}$$

In this two-objective minimization problem, x_1 and x_2 play symmetrical roles in the space of variables, and f_1 and f_2 as well in function space. The test-case raises two major difficulties. Firstly, as depicted in [15], the Pareto front only has very small convex portions present at both of its extremities. The remaining portion, at the center, visually almost occupies the whole front, and it is concave. Recall that the Pareto front associated with convex functions can only be made of convex parts. Thus here the configuration of the front reflects a rupture of convexity of at least one cost-function in the vicinity of most Pareto-optimal solutions. Secondly, the arguments of the above exponential functions are negative, and away from Pareto-optimal points, they become rapidly large enough in magnitude to make the exponential term very close to 0. The same is true for the successive derivatives which involve such exponential terms as factors. As a result, problems occur due to the accuracy of arithmetics. For gradient-based methods particularly, such as the present one, when one gradient nearly vanishes, the risk exists that the numerical method becomes insensitive to the variations of the function, and becomes unable to handle the problem as a two-objective minimization.

In spite of these difficulties, the present method was found able to calculate the Pareto front, in fact very accurately, and over almost the entire range. This was accomplished by choosing two sets of starting points and for each one, an appropriate setting of the method parameters. Before describing the numerical experiments more, the introduction of slack variables and the definition of contraints are made precise.

Here the problem only involves bound constraints on the two unknowns x_1 and x_2. Hence, they can be accounted for by the introduction of two slack variables x_3 and x_4. A simple choice is to let

$$x_1 = 4\sin x_3 \tag{61}$$

$$x_2 = 4\sin x_4 \tag{62}$$

and replace the bound constraints by the following two nonlinear equality constraints:

$$c_1(\mathbf{x}) = x_1 - 4\sin x_3 = 0 \tag{63}$$

$$c_2(\mathbf{x}) = x_2 - 4\sin x_4 = 0. \tag{64}$$

From a feasible starting point, the slack variables are initialized as follows:

$$x_3 = \sin^{-1}\frac{x_1}{4} \tag{65}$$

$$x_4 = \sin^{-1}\frac{x_2}{4} \tag{66}$$

to insure the exact satisfaction of the nonlinear constraints $c_1(\mathbf{x}) = c_2(\mathbf{x}) = 0$ by the starting point. Subsequently, these nonlinear constraints are satisfied by the application of the restoration technique, conditioned by a test of accuracy. Thus, the problem is solved in $\mathbb{R}^4$.

For purpose of evaluation of the accuracy of the numerical results, the Pareto set and front were identified in the following parametric form:

$$x_1(t) = x_2(t) = \frac{t}{\sqrt{2}} \tag{67}$$

$$f_1(t) = 1 - \exp\left[-(t-1)^2\right], \quad f_2(t) = 1 - \exp\left[-(t+1)^2\right] \tag{68}$$

$$-1 \leq t \leq 1. \tag{69}$$

The numerical experiments were conducted for two sets of starting points defined in Table 1.

x_1	x_2
1	0.9
	0.7
	0.5
	0.3
	0.1
-0.01	0.03
	0.10
	0.30
	1.00

Table 1: Initial settings for the experiments on the Fonseca-Fleming test-case

For these nine cases, we have set $\epsilon_0 = 1$, $B = 1$ and $TOL = 10^{-4}$, and performed 25 predictor-corrector iterations in each case by computing the predictor search direction using the MGDA platform [9]. Of course, in this simple problem involving only two cost functions, this search direction could have been calculated by a simple geometrical formula, but the platform was used purposely to demonstrate the operability of all the software elements combined. For purpose of illustration, the Fortran code used in these experiments is provided in Appendix A.

The nine independent convergence paths are shown on Figure 15 in the space of variables, and on Figure 16 in function space. Evidently, the accuracy of the Pareto front definition is excellent. The set of starting points was chosen to obtain Pareto-optimal solutions for which $f_1 < f_2$ in function space. Symmetrical points can be obtained by changing the starting point $\mathbf{x}$ in $-\mathbf{x}$.

The last three figures are related to the convergence of the last case in Table 1 specifically. They illustrate the convergence of the variables (Figure 17), the convergence of the cost-functions (Figure 18), and the convergence of the norm of the ascent vector $\mathbf{d}_\ell^\star$ (Figure 19).

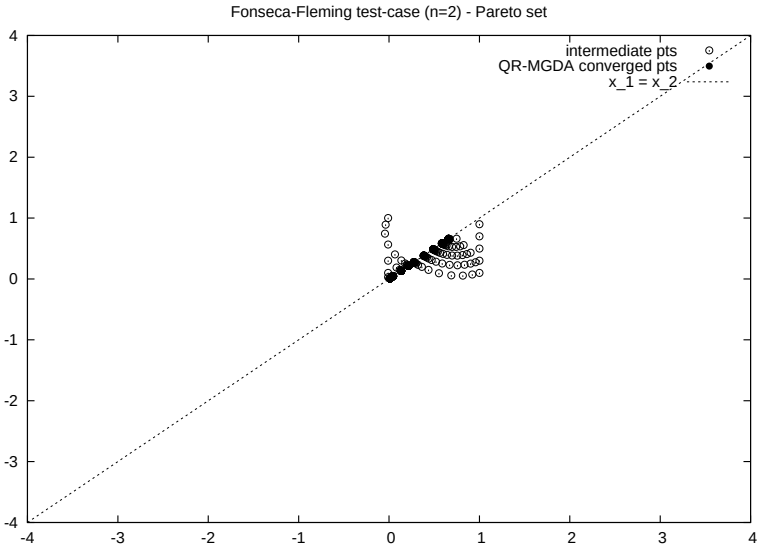


Figure 15: Fonseca-Fleming test-case ($n = 2$). Nine convergence paths to the Pareto set

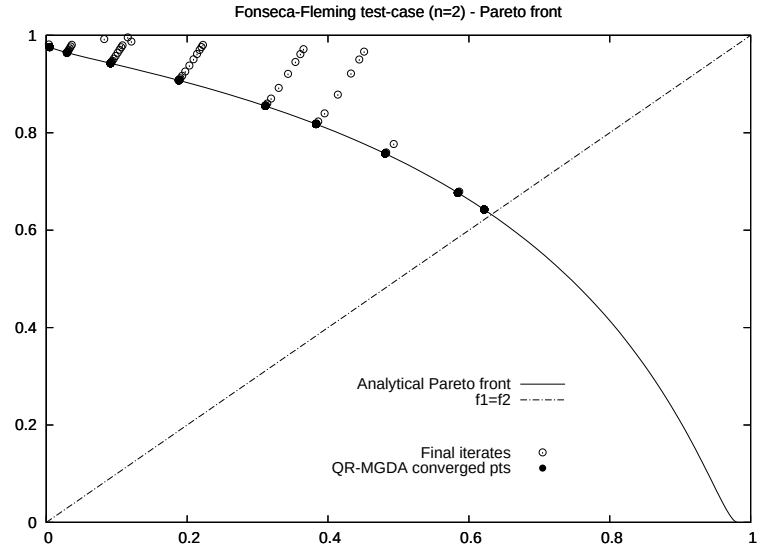


Figure 16: Fonseca-Fleming test-case ($n = 2$). Nine convergence paths to the Pareto front

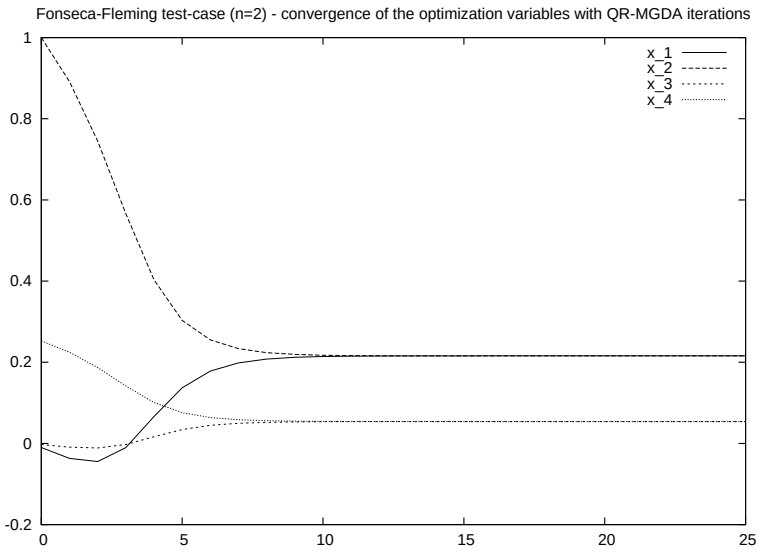


Figure 17: Fonseca-Fleming test-case ($n = 2$). One typical convergence plot for the variables (starting point: $x_1 = -0.01;\ x_2 = 1$)

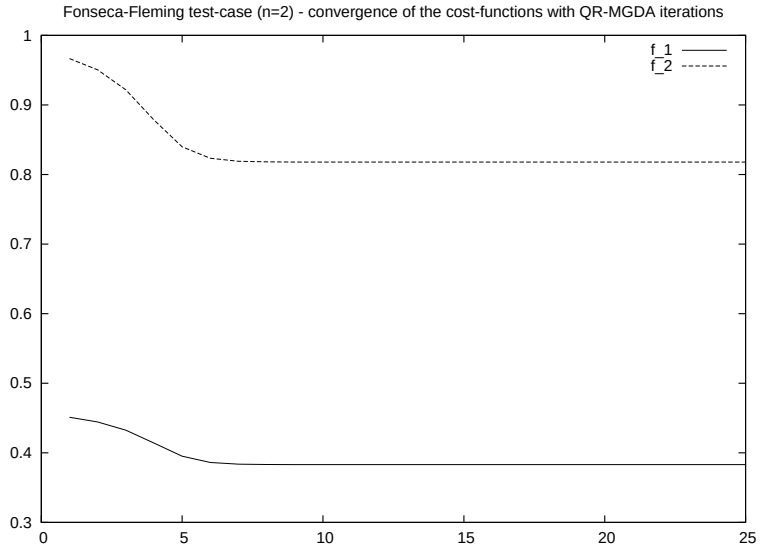


Figure 18: Fonseca-Fleming test-case ($n = 2$). One typical convergence plot for the cost functions (starting point: $x_1 = -0.01;\ x_2 = 1$)

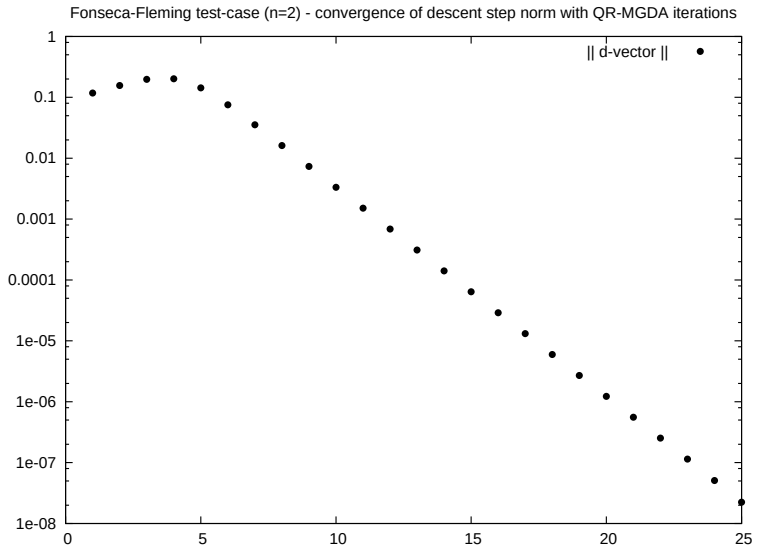


Figure 19: Fonseca-Fleming test-case ($n = 2$). One typical convergence plot for the norm of the ascent vector $\mathbf{d}_\ell^\star$ (starting point: $x_1 = -0.01;\ x_2 = 1$)

7 Conclusion

In this report, the Multiple-Gradient Descent Algorithm (MGDA) for multiobjective differentiable optimization, originally devised for unconstrained problems, has been extended to problems subject to equality constraints, usually nonlinear. The setting encompasses a broad class of situations if slack variables are used. For purpose of demonstration, three optimization test-cases were solved:

1. Problem 47 from the Hock-Schittkowski collection,
2. Constrained minimization of the Rosenbrock function,
3. Fonseca-Fleming two-objective minimization.

Although no special effort was made to establish precisely the domain of convergence of the method in each case, the numerical experimentation has permitted to conclude to the viability of the approach assuming a reasonable starting point is chosen and an appropriate control of the step-size made.

We have proposed to restore the constraints via the minimization of an agglomerated constraint function by an inner-loop quasi-Newton iteration requiring the evaluation of the constraints and constraint gradients only. The efficacy of the procedure has been demonstrated. However, it should be emphasised that it is imperative to update the QR factorization at each inner-loop iteration. Otherwise, the range of the fixed-point process is not the full space, and if the accuracy tolerance has not been achieved at the first iteration, the process may degrade afterwards. However this update is very economical, since the procedure DGEQRF is very efficient.

Additionally, an example Fortran code was provided to demonstrate how can the software be devised to include a call to the *mgdaProcess* available on the MGDA Platform to determine the search direction and solve constrained problems.

References

[1] J.-A. Désidéri, *Multiple-Gradient Descent Algorithm (MGDA)*, Research Report 6953, Inria, June 2009, https://hal.inria.fr/inria-00389811.

[2] ———, *Revision of the Multiple-Gradient Descent Algorithm (MGDA) by Hierarchical Orthogonalization*, Research Report 8710, Inria, April 2015, https://hal.inria.fr/hal-01139994.

[3] ———, *A quasi-Riemannian approach to constrained optimization*, Research Report 9007, Inria, December 2016, https://hal.inria.fr/hal-01417428.

[4] J.-A. Désidéri and R. Duvigneau, *Parametric optimization of pulsating jets in unsteady flow by Multiple-Gradient Descent Algorithm (MGDA)*, Numerical Methods for Differential Equations, Optimization, and Technological Problems, Modeling, Simulation and Optimization for Science and Technology (J. Périaux, W. Fitzgibbon, B. Chetverushkin, and O. Pironneau, eds.), Jan 2017, https://hal.inria.fr/hal-01414741.

[5] C. M. Fonseca and P. J. Fleming, *An Overview of Evolutionary Algorithms in Multiobjective Optimization*, Evolutionary Computation **3** (1995), no. 1, 1–16, DOI:10.1162/evco.1995.3.1.1.

[6] P. E. Gill, W. Murray, and M. H. Wright, *Practical optimization*, Academic Press, London, 1986.

[7] W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems, vol. 187, Springer, http://www.ai7.uni-bayreuth.de/test_problem_coll.pdf.

[8] H.Y. Huang and A.K. Aggerwal, *A class of quadratically convergent algorithms for constrained function minimization*, Journal of Optimization Theory and Applications **16** (1975), no. 5/6, 447–485.

[9] J.-A. Désidéri et al, *MGDA Platform*, March 2017, http://mgda.inria.fr.

[10] Q. Mercier, F. Poirion, and J.-A. Désidéri, *SMGDA : an uncertainty based multi-objective optimization approach. illustration to an airplane composite material*, Procedia Engineering **199** (2017), 1199–1203, https://www.sciencedirect.com/science/article/pii/S1877705817337001.

[11] A. Miele, H.Y. Huang, and J.C. Heideman, *Sequential gradient-restoration algorithm for the minimization of constrained functions, ordinary and conjugate gradient versions*, Journal of Optimization Theory and Applications **4** (1969), no. 4, 213–243.

[12] F. Poirion, Q. Mercier, and J.-A. Désidéri, *Descent algorithm for nonsmooth stochastic multiobjective optimization*, Computational Optimization and Applications **68** (2017), no. 2, 317–331, https://link.springer.com/article/10.1007/s10589-017-9921-x.

[13] P. A. Simionescu and D. Beale, *New Concepts in Graphic Visualization of Objective Functions (PDF), pp. 891-897*, ASME 2002 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (Montreal, Canada), retrieved 7 January 2017.

[14] G. Strang, *Linear algebra and its applications*, 2nd ed., Academic Press, New York London, 1976.

[15] Wikipedia The Free Encyclopedia, *Test functions for optimization*, https://en.wikipedia.org/wiki/Test_functions_for_optimization.

A Fortran code used to generate the results of the Fonseca-Fleming test-case, and one typical formatted output

A.1 Fortran code

The Fortran code used to generate the results of the Fonseca-Fleming test-case is provided here since numerous problems with similar structure could be solved with the same program after only minor modifications. The modifications would consist of a different specification of input data (dimensions, method parameters and starting point) in the main program, and the substitution of specific procedures to 'subroutine FUNS_AND_GRADS' for the calculation of the cost functions and their gradients and 'subroutine CONSTRAINTS' for the calculation of the constraint functions and their gradients.

```
      Program FONSECA

c     This program has been devised to illustrate the application of mgdaProcess
c     from the MGDA Platform http://mgda.inria.fr
c     in a program to solve a multiobjective optimization problem subject to
c     EQUALITY CONSTRAINTS

c     Fonseca-Fleming test-case is specified at :
c     https://en.wikipedia.org/wiki/Test_functions_for_optimization
c     The test-case is here considered with 2 variables (x1,x2) and 2 functions (f1,f2) :
c     f1 (x) = 1 - exp [ - sum_{i=1}^2 (xi-1/sqrt{2})**2 ]
c     f2 (x) = f1(-x)
c     The Pareto front is symmetrical, continuous but not convex except very close to its limits
c     and this is the signature of a rupture of convexity of at least one cost function.

c     The problem is subject to the inequality bounds
c     -4 <= xi <= 4 (i=1,2),
c     that are insured by introducing 2 additional "slack" variables (x3,x4) and
c     the additional nonlinear constraints :
c     xi = 4 * sin x_{i+2]   (i=1,2)
c     Hence the problem is solved in R^4.

c     Starting point(s) : (x1,x2) in [-4,4]
c     x3 = sin^{-1} (x1/4), x4 = sin^{-1}(x2/4)

      use mgdaPilot

      implicit double precision (a-h,o-z)
      parameter (ndimmax=4, mfunmax=2, kcmax=2, lwork=100)

      dimension xv(ndimmax), dstar(ndimmax)
      dimension fun(mfunmax), gradfun(ndimmax,mfunmax)
      dimension pg(ndimmax,mfunmax)
      dimension cfun(kcmax), gradcfun(ndimmax,kcmax), grdgam(ndimmax)

c     Predictor : QRF on contraint gradients
      dimension gdum(ndimmax,kcmax), tau(kcmax), work(lwork)
      dimension qmat(ndimmax,kcmax), rmat(kcmax,kcmax), qtq(kcmax,kcmax)
      dimension scalp(kcmax), xvdum(ndimmax)
      dimension fun0(mfunmax), sigma0(mfunmax), fun1(mfunmax)
```

```
c     Quasi-Newton Corrector : linear system solve
      dimension xvec(kcmax), bvec(kcmax)

c     Formatted output file for main program (fort.1000)
      OPEN (1000, FILE='main-run_report.txt')

c     Formattred output file stecific to mgdaProcess (fort.10)
      OPEN ( 10, FILE='mgda-run_report.txt')
      call mgdaSetOutputDir
     +     ( 50, '/Users/desideri/WDIR/MGDA/MGDAQR-EXP/FONSECA-MGDAP/')


      write (1000,*) 'FONSECA-FLEMING TEST-CASE'
      write (1000,*)
     s 'BY MGDA SUBJECT TO EQUALITY CONSTRAINTS ON SLACK VARIABLES'

c     Enter dimensions:
      ndim = 4
      write (1000,*)
      write (1000,*)
     s     'Dimension of optimization variable xv, ndim = ', ndim

      mfun = 2
      write (1000,*) 'Number of cost-functions, mfun = ', mfun

      kc = 2
      write (1000,*) 'Number of equality constraints, kc = ', kc

c     Enter MGDA options
      write (6,*)
      write (6,*) 'MGDA options :'
      write (1000,*)
      write (1000,*) 'MGDA options :'
ccc      write (6,*) 'Assign iscale :'
ccc      read (5,*) iscale
      iscale = 0
      write (1000,*) 'iscale = ', iscale
ccc      write (6,*) 'Assign logmode :'
ccc      read (5,*) logmode
      logmode = 0
      write (1000,*) 'logmode = ', logmode
ccc      write (6,*) 'Assign first reference stepsize eps0 :'
ccc   read (5,*) eps0
      eps0 = 1.d0
      write (1000,*) 'First reference step-size, eps0 = ', eps0
ccc      write (6,*)
ccc     1 'Assign maximum number of iterations, itermax :'
ccc   read (5,*) itermax
      itermax = 25
      write (1000,*) 'Maximum number of MGDA iterations, itermax :',
     1     itermax
      eps_Hdiag = 1.d-10

c     Enter options for the corrector iteration
```

```
      write (6,*)
      write (6,*) 'Options for the quasi-Newton corrector iteration :'
      write (1000,*)
      write (1000,*)
     + 'Options for the quasi-Newton corrector iteration :'
ccc      write (6,*) 'Assign constraint-violation tolerance, TOL :'
ccc   read (5,*) TOL
      TOL = 1.d-4
      write (1000,*) 'Accuracy tolerance on constraints, TOL :', TOL
ccc      write (6,*) 'Assign bound on constraint violation, boundc :'
ccc   read (5,*) boundc
      boundc = 1.d-2
      write (1000,*) 'Bound on constraint violation, boundc = ', boundc
ccc      write (6,*)
ccc     s 'Assign allowable number of Newton_s iterations, maxiter :'
ccc   read (5,*) maxiter
      maxiter = 4
      write (1000,*)
     s  'Allowable number of Newton_s iterations, maxiter = ', maxiter

      OPEN(120,ACCESS='APPEND')
      OPEN(121,ACCESS='APPEND')
      OPEN(130,ACCESS='APPEND')
      OPEN(131,ACCESS='APPEND')

c     Initialize optimization vector xv
      write (6,*)
      write (6,*)
     s 'Assign values in [-4,4] to x1 and x2 :'
      read (5,*) xv(1), xv(2)
      xv(3) = asin ( xv(1)/4.d0 )
      xv(4) = asin ( xv(2)/4.d0 )
      write (1000,*)
      write (1000,*) 'Starting point xv :'
      do i = 1,ndim
         write (1000,*) xv(i)
      end do
      write (12,*) 0, (xv(i), i=1,ndim)
      write (120,*) 0, (xv(i), i=1,ndim)

      write (1000,*)
      write (1000,*) 'Enter MGDA optimization loop'

      iter = 0
 1    iter = iter + 1
      write (1000,*)
      write (1000,*) '============================================='
      write (1000,*) 'Iteration ', iter
      write (1000,*)
      write (1000,*)
     s    'MGDA PREDICTOR STEP SUBJECT TO LINEARIZED CONSTRAINTS :'

c     compute values of functions and gradients
      call FUNS_AND_GRADS (xv, ndim, fun, mfun, gradfun, ndimmax)
```

```
c     store function values
      do j = 1,mfun
         fun0(j) = fun(j)
      end do

      write (13,*) iter, (fun(j), j=1,mfun)
      write (130,*) iter, (fun(j), j=1,mfun)

      write (1000,*)
      write (1000,*) 'Cost-function values :'
      write (1000,100) (fun(j), j=1,mfun)
 100  format(6F15.6)
      write (1000,*)
      write (1000,*) 'Cost-function gradients :'
      do i = 1,ndim
         write (1000,100) (gradfun(i,j), j =1,mfun)
      end do

c     compute constraint values (should be =0) and constraint gradients
      call CONSTRAINTS
     1 (xv, ndim, cfun, cmax, gradcfun, ndimmax, kc, gamma, grdgam)
      write (1000,*)
      write (1000,*) 'Constraint values :'
      write (1000,*) (cfun(k), k=1,kc)
      write (1000,*)
      write (1000,*) 'Constraint gradients :'
      do i = 1,ndim
         write (1000,*) (gradcfun(i,k), k=1,kc)
      end do

      iverbose = 1
      call QRFACTORY ( gradcfun, gdum, ndim, ndimmax, kc, kcmax,
     s work, lwork, tau, qmat, rmat, qtq, xvec, bvec, iverbose)

c     compute projected gradients pg(ndim,mfun) as follows :
c     for j = 1,mfun
c     pg(*,j) = gradfun(*,j) - sum_k (gradfun(*,j),qmat(*,k)) * qmat(*,k), k = 1,kc
      write (1000,*)
      write (1000,*) 'Vector pg(*,j) = gradfun(*,j) - '
     1     , 'sum_k (gradfun(*,j),qmat(*,k)) * qmat(*,k), k = 1,kc'

      do j = 1,mfun

         do k = 1,kc
            scalp(k) = 0.d0
            do i = 1,ndim
               scalp(k) = scalp(k) + gradfun(i,j) * qmat(i,k)
            end do
         end do

         do i = 1,ndim
            pg(i,j) = gradfun(i,j)
            do k = 1,kc
```

```
                pg(i,j) = pg(i,j) - scalp(k) * qmat(i,k)
             end do
          end do

       end do
       write (1000,*)
       write (1000,*) 'Projected gradients (pg) :'
       do i = 1,ndim
          write (1000,100) (pg(i,j), j=1,mfun)
       end do

c      LOAD FORT.99 TO BE READ BY mgdaProcess
       OPEN (99)
       write (99,101) 'Fonseca-Fleming Test-Case'
 101   format(a50)
       write (99,*) mfun
       write (99,*) ndim
       write (99,*)
       do j = 1,mfun
          write (99,*) j
          write (99,*) fun(j)
          do i =1,ndim
             write (99,*) pg(i,j)
          end do
       end do
       write (99,*)
       write (99,*) 'Iteration ', iter
       CLOSE (99)

c      compute descent direction by MGDA
c      compute dstar
       call mgdaProcess (ndim, mfun, fun, pg,
      1 logmode, iscale, eps_Hdiag, dstar)

       write (1000,*)
       write (1000,*) 'Ascent vector dstar :'
       dnorm = 0.d0
       do i = 1,ndim
          write (1000,*) dstar(i)
          dnorm = dnorm + dstar(i)**2
       end do
       dnorm = sqrt(dnorm)
       write (1000,*)
       write (1000,*) 'Norm of dstar = ', dnorm
       write (11,*) iter, dnorm

       write (1000,*)
       write (1000,*) 'Directional derivatives : '
       do j = 1,mfun
          sigj = 0.d0
          do i = 1,ndim
             sigj = sigj + gradfun(i,j) * dstar(i)
          end do
          sigma0(j) = sigj
```

```
      end do
      write (1000,100) (sigma0(j), j=1,mfun)

c     PREDICTOR STEP :
c     Step-size selection
 3    write (1000,*)
      write (1000,*) 'Step-size selection :'
      write (1000,*)
      write (1000,*) 'Initial setting : eps = eps0 = ', eps0
      eps1 = eps0
c     1. Constraint violation limitation
      write (1000,*) '1. Constraint violation limitation :'
c     xv0 :
      do i = 1,ndim
         xvdum(i) = xv(i) - eps0 * dstar(i)
      end do
      call CONSTRAINTS
     1  (xvdum, ndim, cfun, cmax, gradcfun, ndimmax, kc, gamma, grdgam)
      write (1000,*) 'cmax0 = ', cmax
      eps1 = eps0 * sqrt( boundc / cmax )
      eps1 = min ( eps0, eps1)
      write (1000,*) 'eps1 = ', eps1
c     2. Optimization of eps in [0,eps1]
      write (1000,*) '2. Optimization of eps in [0,eps1]'
c     xv1 :
      do i = 1,ndim
         xvdum(i) = xv(i) - eps1 * dstar(i)
      end do
      call FUNS_AND_GRADS (xvdum, ndim, fun1, mfun, gradfun, ndimmax)
      epsstar = eps1
      do j = 1,mfun
         delj = fun1(j) - fun0(j) + sigma0(j) * eps1
         if(delj.gt.0) then
            epsbarj = 0.5d0 * sigma0(j) * eps1**2 / delj
            if(epsbarj.lt.epsstar) epsstar = epsbarj
         end if
      end do
      if(epsstar.eq.eps0) then
         eps0 = 2.d0 * eps0
         write (1000,*) 'eps0 doubled : eps0 = ', eps0
         go to 3
      end if
c     Definite step-size
      write (1000,*) 'Definite step-size, epsstar = ', epsstar
c     Predictor update
      do i = 1,ndim
         xv(i) = xv(i) - epsstar * dstar(i)
      end do

      write (1000,*)
      write (1000,*) 'Updated point xv (end of MGDA predictor step):'
      do i = 1,ndim
         write (1000,*) xv(i)
      end do
```

```
      write (12,*) iter, (xv(i), i =1,ndim)
      write (120,*) iter, (xv(i), i =1,ndim)

      write (1000,*)
      write (1000,*) 'CONSTRAINT-RESTORATION CORRECTOR STEP :'

      newtoniter = 0
 2    call CONSTRAINTS
     s     (xv, ndim, cfun, cmax, gradcfun, ndimmax, kc, gamma, grdgam)
      write (1000,*)
      write (1000,*) 'Quasi-Newton iteration ', newtoniter,
     s '   Constraint values and cmax :'
      write (1000,*) (cfun(k), k=1,kc), '   cmax = ', cmax

      if(cmax.gt.tol) then
         if(newtoniter.le.maxiter) then

            iverbose = 0

            call QRFACTORY ( gradcfun, gdum, ndim, ndimmax, kc, kcmax,
     s         work, lwork, tau , qmat, rmat, qtq, xvec, bvec, iverbose)

            call QUASI_NEWTON_ITERATION
     s         ( xv, ndim, ndimmax, cfun, kc, kcmax, qmat, rmat, xvec)

         end if
         newtoniter = newtoniter + 1
         if(newtoniter.lt.maxiter) go to 2
      end if

      write (1000,*)
      write (1000,*) 'Corrected solution xv at iteration ', iter
      do i = 1,ndim
         write (1000,100) xv(i)
      end do

      if(iter.lt.itermax) GO TO 1

      write (1000,*)
      write (1000,*) 'Cost-function values :'
      call FUNS_AND_GRADS (xv, ndim, fun, mfun, gradfun, ndimmax)
      write (1000,100) (fun(j), j=1,mfun)

      write (121,*) iter, (xv(i), i =1,ndim)
      write (131,*) iter, (fun(j), j=1,mfun)

      END

      subroutine FUNS_AND_GRADS (xv, ndim, fun, mfun, grad, ndimmax)
      implicit double precision (a-h,o-z)
      dimension xv(*), fun(*), grad(ndimmax,*)

c     Fonseca-Fleming test-case : 2 optimization variables and 2 slack variables (ndim=4)
```

```
      sqr = sqrt(0.5d0)

      y1 = xv(1)-sqr
      y2 = xv(2)-sqr
      exp1 = exp( -y1**2 -y2**2 )

      z1 = xv(1)+sqr
      z2 = xv(2)+sqr
      exp2 = exp( -z1**2 -z2**2 )

      fun(1) = 1.d0 - exp1
      fun(2) = 1.d0 - exp2

      grad(1,1) = 2.d0 * y1 * exp1
      grad(2,1) = 2.d0 * y2 * exp1
      grad(3,1) = 0.d0
      grad(4,1) = 0.d0

      grad(1,2) = 2.d0 * z1 * exp2
      grad(2,2) = 2.d0 * z2 * exp2
      grad(3,2) = 0.d0
      grad(4,2) = 0.d0


      RETURN
      END

      subroutine CONSTRAINTS
     1 (xv, ndim, cfun, cmax, gradcfun, ndimmax, kc, gamma, grdgam)
      implicit double precision (a-h,o-z)
      dimension xv(ndim), cfun(kc), gradcfun(ndimmax,kc), grdgam(ndim)

c     Fonseca-Fleming test-case n=kc=2 ndim=4; (x3,x4) : slack variables
c     c_i(x) = x_i - 4 sin(x_{i+2}/4)

c     values :
      do i = 1,kc
         cfun(i) = xv(i) - 4.d0 * sin( xv(i+kc) )
      end do

c     compute maximum constraint violation cmax
      cmax = abs(cfun(1))
      do k = 2,kc
         test = abs(cfun(k))
         if(test.gt.cmax) cmax = test
      end do

c     gradients
      do k = 1,kc
         do i = 1,ndim
            gradcfun(i,k) = 0.d0
         end do
         gradcfun(k,k) = 1.d0
```

```
         gradcfun(k+kc,k) = -4.d0 * cos( xv(k+kc) )
      end do

c     function gamma and gradient
      gamma = 0.d0
      do k = 1,kc
         ck = cfun(k)
         gamma = gamma + ck**2
         do i = 1,ndim
            grdgam(i) = grdgam(i) + ck * gradcfun(i,k)
         end do
      end do
      gamma = 0.5d0 * gamma

      RETURN
      END

      subroutine QRFACTORY ( gradcfun, gdum, ndim, ndimmax, kc, kcmax,
     s  work, lwork, tau , qmat, rmat, qtq, xvec, bvec, iverbose)

      implicit double precision (a-h,o-z)
      dimension gradcfun(ndimmax,kc), gdum(ndimmax,kc)
      dimension work(lwork), tau(kc)
      dimension qmat(ndimmax,kc), rmat(kcmax,kc), qtq(kcmax,kc)
      dimension xvec(kc), bvec(kc)

 100  format(10F10.6)

c     apply QR-factorization to constraint gradients
c     load duplicate of constraint gradients
      do i = 1,ndim
         do k = 1,kc
            gdum(i,k) = gradcfun(i,k)
         end do
      end do
c     perform QRF

      call DGEQRF (ndim, kc, gdum, ndimmax, tau, work, lwork, info)
c     DGEQRF( M, N, A, LDA, TAU, WORK, LWORK, INFO )

      if(info.ne.0) then
         write (1000,*)
         write (1000,*) 'Upon exit of DGEQRF, info = ', info
      end if

c     read Rmat in Gdum
      if(iverbose.gt.0) then
         write (1000,*)
         write (1000,*) 'Matrix Rmat :'
      end if
      do i = 1,kc
         do j = 1,i-1
            rmat(i,j) = 0.d0
         end do
```

```
         do j = i,kc
            rmat(i,j) = gdum(i,j)
         end do
         if(iverbose.gt.0) write (1000,100) (rmat(i,j), j=1,kc)
      end do

c     solve ndim linear systems R^t X = Gradcfun^t to to get Q = X^t

      do i = 1,ndim
         do k = 1,kc
            bvec(k) = gradcfun(i,k)
         end do
         call RTX_SOLVE ( rmat, kc, kcmax, xvec, bvec)
         do k = 1,kc
            qmat(i,k) = xvec(k)
         end do
      end do

       if(iverbose.gt.0) then

            write (1000,*)
            write (1000,*) 'Matrix Qmat :'
            do i = 1,ndim
               write (1000,100) (qmat(i,j), j=1,kc)
            end do

c           4: verification of orthonormality:
            write (1000,*)
            write (1000,*) 'Matrix Qmat-transpose * Qmat :'
            do i = 1,kc
               do j = 1,kc
                  qtqij = 0.d0
                  do l = 1,ndim
                     qtqij = qtqij + qmat(l,i) * qmat(l,j)
                  end do
                  qtq(i,j) = qtqij
               end do
               write (1000,100) (qtq(i,j), j=1,kc)
            end do

         end if
c        (end of verbose)

      RETURN
      END

      subroutine QUASI_NEWTON_ITERATION
     s ( xv, ndim, ndimmax, cfun, kc, kcmax, qmat, rmat, xvec)

      implicit double precision (a-h,o-z)
      dimension xv(ndim)
      dimension cfun(kc), xvec(kc)
      dimension qmat(ndimmax,kc), rmat(kcmax,kc)
```

```
c     solve system Rmat^t * xvec = cfun by forward substitution, and set eta = -xvec
      call RTX_SOLVE ( rmat, kc, kcmax, xvec, cfun)

c     update xv
      do i = 1,ndim
         do k = 1,kc
            dxvi = -xvec(k) * qmat(i,k)
            xv(i) = xv(i) + dxvi
         end do
      end do

      RETURN
      END

      subroutine RTX_SOLVE ( rmat, kc, kcmax, xvec, bvec)

      implicit double precision (a-h,o-z)
      dimension xvec(kc), bvec(kc)
      dimension rmat(kcmax,kc)

c     solve system Rmat^t * xvec = bvec by forward substitution :
      xvec(1) = bvec(1) / rmat(1,1)
      do i = 2,kc
         sum = 0.d0
         do k = 1,i-1
            sum = sum + rmat(k,i) * xvec(k)
         end do
         xvec(i) = ( bvec(i) - sum) / rmat(i,i)
      end do

      RETURN
      END
```

A.2 One typical formatted output 'main-run_report.txt' (starting point $x_1 = -0.01;\ x_2 = 1$) (1st and 25th iterations only)

```
 FONSECA-FLEMING TEST-CASE
 BY MGDA SUBJECT TO EQUALITY CONSTRAINTS ON SLACK VARIABLES

 Dimension of optimization variable xv, ndim =            4
 Number of cost-functions, mfun =            2
 Number of equality constraints, kc =            2

 MGDA options :
 iscale =            0
 logmode =            0
 First reference step-size, eps0 =    1.0000000000000000
 Maximum number of MGDA iterations, itermax :          25

 Options for the quasi-Newton corrector iteration :
 Accuracy tolerance on constraints, TOL :   1.0000000000000000E-004
 Bound on constraint violation, boundc =    1.0000000000000000E-002
 Allowable number of Newton_s iterations, maxiter =            4
```

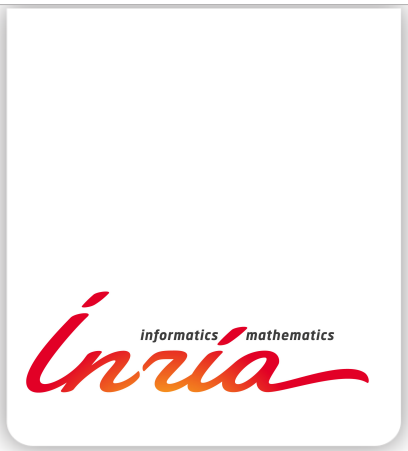
```
Starting point xv :
 -1.0000000000000000E-002
  1.0000000000000000
 -2.5000026041739911E-003
 0.25268025514207865

Enter MGDA optimization loop

============================================
Iteration            1

MGDA PREDICTOR STEP SUBJECT TO LINEARIZED CONSTRAINTS :

Cost-function values :
      0.451204       0.966632

Cost-function gradients :
     -0.787091       0.046521
      0.321477       0.113924
      0.000000       0.000000
      0.000000       0.000000

Constraint values :
  0.0000000000000000        0.0000000000000000

Constraint gradients :
  1.0000000000000000        0.0000000000000000
  0.0000000000000000        1.0000000000000000
 -3.9999874999804685        0.0000000000000000
  0.0000000000000000       -3.8729833462074170

Matrix Rmat :
-4.123093  0.000000
 0.000000 -4.000000

Matrix Qmat :
-0.242536 -0.000000
-0.000000 -0.250000
 0.970142 -0.000000
-0.000000  0.968246

Matrix Qmat-transpose * Qmat :
 1.000000  0.000000
 0.000000  1.000000

Vector pg(*,j) = gradfun(*,j) - sum_k (gradfun(*,j),qmat(*,k)) * qmat(*,k), k = 1,kc

Projected gradients (pg) :
     -0.740791       0.043785
      0.301385       0.106804
     -0.185198       0.010946
      0.077817       0.027577

Ascent vector dstar :
```

```
  2.7591956813400896E-002
 0.11081967946370336
  6.8980107596675281E-003
  2.8613518199664491E-002

Norm of dstar =   0.11793486937048582

Directional derivatives :
      0.013909       0.013909

Step-size selection :

Initial setting : eps = eps0 =    1.0000000000000000
1. Constraint violation limitation :
cmax0 =    3.9421743889223038E-004
eps1 =    1.0000000000000000
2. Optimization of eps in [0,eps1]
Definite step-size, epsstar =   0.97729595695519778

Updated point xv (end of MGDA predictor step):
 -3.6965507838219121E-002
 0.89169637530905177
 -9.2414006306305186E-003
 0.22471637949128256

CONSTRAINT-RESTORATION CORRECTOR STEP :

Quasi-Newton iteration            0    Constraint values and cmax :
 -4.3147866745502084E-007   3.7684906302615229E-004    cmax =    3.7684906302615229E-004

Quasi-Newton iteration            1    Constraint values and cmax :
 -1.8735013540549517E-016   3.6649691059764677E-009    cmax =    3.6649691059764677E-009

Corrected solution xv at iteration            1
     -0.036965
      0.891673
     -0.009242
      0.224807

================================================
Iteration           25

MGDA PREDICTOR STEP SUBJECT TO LINEARIZED CONSTRAINTS :

Cost-function values :
      0.383009       0.817911

Cost-function gradients :
     -0.606349       0.336077
     -0.606349       0.336077
      0.000000       0.000000
      0.000000       0.000000

Constraint values :
```

```
  1.1057977894274318E-005   2.5429560902817672E-005

Constraint gradients :
  1.0000000000000000        0.0000000000000000
  0.0000000000000000        1.0000000000000000
 -3.9941788762081871        0.0000000000000000
  0.0000000000000000       -3.9941796497560946

Matrix Rmat :
-4.117459  0.000000
 0.000000 -4.117459

Matrix Qmat :
-0.242868 -0.000000
-0.000000 -0.242868
 0.970059 -0.000000
-0.000000  0.970059

Matrix Qmat-transpose * Qmat :
 1.000000  0.000000
 0.000000  1.000000

Vector pg(*,j) = gradfun(*,j) - sum_k (gradfun(*,j),qmat(*,k)) * qmat(*,k), k = 1,kc

Projected gradients (pg) :
     -0.570584       0.316254
     -0.570584       0.316254
     -0.142854       0.079179
     -0.142854       0.079179

Ascent vector dstar :
 -1.5364214485828601E-008
  1.5364215122729784E-008
 -3.8466515820155171E-009
  3.8466509909159522E-009

Norm of dstar =    2.2398920628981238E-008

Directional derivatives :
      0.000000       0.000000

Step-size selection :

Initial setting : eps = eps0 =    1.0000000000000000
1. Constraint violation limitation :
cmax0 =    2.5429560902817672E-005
eps1 =    1.0000000000000000
2. Optimization of eps in [0,eps1]
Definite step-size, epsstar =   0.85832763871432216

Updated point xv (end of MGDA predictor step):
 0.21573103976430524
 0.21573106181243082
  5.3956171786309619E-002
```

```
  5.3952579174676996E-002

 CONSTRAINT-RESTORATION CORRECTOR STEP :

 Quasi-Newton iteration            0    Constraint values and cmax :
   1.1057977894246562E-005   2.5429560902817672E-005    cmax =    2.5429560902817672E-005

 Corrected solution xv at iteration           25
       0.215731
       0.215731
       0.053956
       0.053953

 Cost-function values :
       0.383009       0.817911
```

Contents



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399