Computational Optimization and Applications, 15, 193–205 (2000)

# An Implementation of Shor's *r*-Algorithm

FRANZ KAPPEL franz.kappel@kfunigraz.ac.at
ALEXEI V. KUNTSEVICH alex@bedvgm.kfunigraz.ac.at
*Institute for Mathematics, University of Graz, Heirichstr., 36, A-8010 Graz, Austria*

*Received April 7, 1998; Accepted January 6, 1999*

**Abstract.** Here we introduce a new implementation of well-known Shor's $r$-algorithm with space dilations along the difference of two successive (sub)gradients for minimization of a nonlinear (non-smooth) function (N.Z. Shor, Minimization methods for Non-Differentiable Functions, Springer-Verlag: Berlin, 1985. Springer Series in Computational Mathematics, vol. 3). The modifications made to Shor's algorithm are heuristic. They mostly concern the termination criteria and the line search strategy. A large number of test runs indicate that this implementation is very robust, efficient and accurate. We hope that this implementation of Shor's $r$-algorithm will prove to be useful for solving a wide class of non-smooth optimization problems.

**Keywords:** nonsmooth, nondifferential optimization, nonlinear programming

## 1. Introduction

Shor's $r$-algorithm seems to be a very efficient method for minimization of non-smooth (i.e., almost differentiable) functions. However, a serious problem for Shor's algorithm is the design of an efficient stopping criterion, a difficulty common to all algorithms for optimization of non-smooth functions. A further difficulty is the choice of the initial step size. Below we describe in detail the modifications and the additions which resulted in a robust and efficient algorithm.

We start with a description of Shor's original algorithm with space dilation along the difference of two successive subgradients. In Section 3 we present in detail the main features of our implementation. Main emphasis is on those parts which had to be added in order to get a robust and efficient algorithm. Finally, in Section 5 we give some information on the program and on the performance of the algorithm.

## 2. Shor's *r*-algorithm

The main idea of the algorithm is to make steps in the direction opposite to a sub-gradient at the current point. However, the steps are to be made in the transformed space. The way to perform this transformation is quite simple. At each iteration one calculates the difference between a subgradient at the current point and that calculated at the previous step. The direction obtained is used to perform dilation of the space with a priori given coefficient. For a more detailed description we refer to [6].

Let $f(\cdot)$ be an almost differentiable convex function defined on $\mathbb{R}^n$ which is differentiable on its domain except on a set of measure zero. Let us denote an almost gradient of $f(\cdot)$ at the point $x$ by $g_f(x)$. Consider the following iterative algorithm for minimization of the function $f(\cdot)$.

At the initial step, compute a subgradient $g_f(x_0)$ for a given starting point $x_0$, choose $h_1 > 0$ and find $x_1 = x_0 - h_1 g_f(x_0)$, set $\tilde{g}_1 = g_f(x_0)$ and $B_0 = I$ (the unit $n \times n$ matrix).

Assume that after $k$ iterations one has obtained the point $x_k$, the space transformation matrix $B_k$ and the subgradient $\tilde{g}_k$ of the function $\varphi_k(y) = f(B_k y)$ at the point $\tilde{y}_k = B_k^{-1} x_{k-1}$.

At the $(k+1)$-st iteration the following calculations have to be performed:

1. Calculate $g_f(x_k)$, a subgradient of $f$ at $x_k$.
2. Calculate $g_k^* = B_k^T g_f(x_k)$, a subgradient of $\varphi_k$ at the point $y_k = B_k^{-1} x_k$.
3. Calculate $r_k = g_k^* - \tilde{g}_k$, the difference of the two subgradients of $\varphi_k$ at $y_k$ and $\tilde{y}_k$.
4. Set $\xi_{k+1} = r_k / \|r_k\|$. The normalized vector $\xi_{k+1}$ is the direction of the next space dilation to be performed.
5. Calculate $B_{k+1} = B_k R_\beta(\xi_{k+1})$, where $\beta = 1/\alpha$, $\alpha > 1$ is a fixed constant. The matrix $R_\beta(\xi_{k+1})$ is the inverse of $R_\alpha(\xi_{k+1})$, the matrix of the space dilation in the direction $\xi_{k+1}$ with coefficient $\alpha$ given by

$$R_\alpha(\xi_{k+1})x = x + (\alpha - 1)(x^T \xi_{k+1})\xi_{k+1}, \quad x \in \mathbb{R}^n.$$

6. Calculate $\tilde{g}_{k+1} = B_{k+1}^T g_f(x_k)$, a subgradient of the function $\varphi_{k+1}(y) = f(B_{k+1} y)$ at the point $\tilde{y}_{k+1} = B_{k+1}^{-1} x_k$.
7. Choose a step size $h_{k+1}$.
8. Set $x_{k+1} = x_k - h_{k+1} B_{k+1} \tilde{g}_{k+1}$.
9. Check the stopping criterion and stop if it is satisfied. Otherwise proceed to the next iteration.

## 3. Modifications to Shor's algorithm

To turn the $r$-algorithm as presented above into an efficient and robust optimization routine, one has to find solutions to the following problems:

- Initialization and re-initialization of the space transformation matrix $B_k$ and initialization of the step size $h_k$.
- Choice of the step size $h_{k+1}$ to optimize the efficiency of space dilation in the direction of the difference of two successive subgradients (of the transformed function $\varphi_k$).
- Construction of a stopping criterion which does not need information on gradients.

In the following subsections we describe the solutions to these problems implemented in *SolvOpt* (*Solv*er for local nonlinear *Opt*imization problems). The solutions are heuristic.

### *3.1. Initial trial step size*

The stepsize $\hat{h}_1$ should not be too small if a function is flat, and not too large if a function is steep. How can we satisfy these requirements, having no other information on the objective function but the starting point and the gradient calculated at this point? In view of these conditions and our limited knowledge the following formula is suggested for the first trial step along the gradient calculated at the starting point:

$$\hat{h} = \frac{c}{\log_2(\|g_0\|_2 + 1)}, \tag{1}$$

where $c > 0$ is a factor which can be chosen. The default value 1.

### *3.2. Re-initialization*

The space transformation matrix reflects the history of the optimization process. It should cause the algorithm to make steps mainly towards the optimum instead of making steps along the current gradient, which may be almost orthogonal to the desired direction. A similar strategy is pursued in quasi-Newton methods. However, occasionally one has to forget the history of the optimization process. Since the space dilation coefficient is larger than 1, the norm of the gradients in the transformed space decreases from iteration to iteration. If the dilations occur more or less in the same direction, which is the most preferable situation, then the norm of the transformed gradients possibly becomes close to zero only near the optimum. However, if space dilations are repeatedly made in different directions which almost give an orthogonal basis of the space, then the norm of the transformed gradients may become very small also far away from the optimum. In this case, it is advisable to start the algorithm anew at the best point obtained so far. Such a situation can occur for ravine shaped functions (e.g. the Shell Dual Problem [4] and for functions which are badly scaled. The problem is to decide when one should reset the algorithm, so that it starts anew at the present point and the past history of the process does not interfere in choosing the next direction to go. The following simple criterion is based on heuristics and proved to be efficient:

*Resetting of the space dilation matrix*:
If at the $k$th iteration the inequality

$$\|\tilde{g}_k\|_2 \le 10^{-15}\|g_k\|_2$$

is satisfied, where $g_k$, $\tilde{g}_k$ are the gradients of the functions $f(x)$ and $\varphi(y)$, then set $B_{k+1} = I$, i.e., the space transformation at the next step is the identity transformation.

### *3.3. The step size strategy*

Any optimization algorithm which uses the gradient of the objective function invokes at each iteration a line search procedure to determine the step size. In algorithms for smooth problems, the line search procedure usually tries to find or to approximate the optimizer of the objective function $f$ in the chosen direction. However, in Shor's $r$-algorithm it seems to be necessary to find a step size such that the optimizer for $f$ in the chosen direction is a point between the current point and the next point of the solution path. This is the most efficient way to guarantee that the direction for space dilation at the next step (the difference of the gradients of $\varphi$ at $y_k$ and $\tilde{y}_k$) points in the direction where the valley of the objective function which just has been crossed descends to a local minimum or ascends to a local maximum. The step size strategy implemented in *SolvOpt* also avoids increasing the objective function too much in one step. However, the objective function might increase occasionally, so the algorithm is not a true descent algorithm. This fact, together with the difficulty of keeping track of the successive space transformations, is the main reason that at present there does not exist a sufficiently general proof of convergence for the $r$-algorithm.

In the description below it is understood that the task is to find a local minimum of the objective function. We assume that steps 1–6 of the $(k+1)$-st iteration of the $r$-algorithm have been completed, so that $x_k$, $g_k$, $\tilde{g}_{k+1}$ and $B_{k+1}$ are already computed. Moreover, the trial step size $\hat{h}_{k+1}$ for the line search in the $(k+1)$-st iteration has also been determined. Then we have to perform the following calculations:

a) Set $j_{k+1} = 0$, $h_{k+1} = \hat{h}_{k+1}$, $x_{k+1}^{(0)} = x_k$ and $f^{(0)} = f(x_{k+1}^{(0)})$.
b) Calculate

$$\gamma = \min\left\{1 + b_1^{(p-\tau)n}, b_2^{\max(\log_{10}(\|g_k\|+1),1)}\right\}, \tag{2}$$

where $b_1 = 1 + 1/(10n^2)$, $b_2 = 1.15$ and $p = 10$ are the chosen a priori constants, $\tau$ is the number of iterations made after the last resetting. By calculating $\gamma$ the way it is done in (2), one specifies a new, more strict, upper bound on an occasional increase of the function value every next iteration, hence brings the next point closer to the line search minimizer.
c) Calculate $x_{k+1}^{(j_{k+1}+1)} = x_{k+1}^{(j_{k+1})} - h_{k+1}B_{k+1}\tilde{g}_{k+1}$, $f^{(j_{k+1}+1)} = f(x_{k+1}^{(j_{k+1}+1)})$.
d) Set $j_{k+1} = j_{k+1} + 1$.
e) If $f^{(j_{k+1})} \geq \gamma f^{(j_{k+1}-1)}$, then set $h_{k+1} = h_{k+1}/5.1$, $j_{k+1} = 0$ and continue with c) (the used stepsize is too large). Otherwise go to f).
f) If $f^{(j_{k+1})} < f^{(j_{k+1}-1)}$, check the step counter. Set

$$h_{k+1} = \begin{cases} 2h_{k+1}, & \text{if } j_{k+1} > 20,\\ 1.5h_{k+1}, & \text{if } 20 \geq j_{k+1} > 10,\\ 1.05h_{k+1}, & \text{if } 10 \geq j_{k+1} > 2,\\ h_{k+1}, & \text{if } j_{k+1} \leq 2 \end{cases}$$

and continue with c). Otherwise go to g).
g) Set $x_{k+1} = x_{k+1}^{(j_{k+1}+1)}$ and $\tilde{j} = (j_{k-1} + 2j_k + 3j_{k+1})/6$ assuming $k \geq 2$. Check the value

$\tilde{j}$ and set

$$\tilde{h}_{k+2} = \begin{cases} \sqrt{\tilde{j} - j_0 + 1}\, \tilde{h}_{k+1}, & \text{if } \tilde{j} > j_0, \\ \tilde{h}_{k+1}, & \text{if } \tilde{j} = j_0, \\ \sqrt{\tilde{j}/j_0}\, \tilde{h}_{k+1}, & \text{if } \tilde{j} < j_0, \end{cases}$$

where $j_0 = 3.3$, if the gradients are calculated analytically and $j_0 = 6.3$, otherwise.

Note that item g) is reached only if $f(x_{k+1}^{(j_{k+1})}) \geq f(x_{k+1}^{(j_{k+1}-1)})$.

### *3.4. Termination*

The stopping criterion used in *SolvOpt* (besides the condition that the maximum number of iterations is not exceeded) can only be seen in connection with the step size strategy implemented in *SolvOpt*. The efficiency of the stopping criterion can be explained by the fact that, in general, the minimum (maximum) of the objective function on the line joining $x_k$ and $x_{k+1}$ is attained at a point between $x_k$ and $x_{k+1}$. This prevents the step size from becoming too small if $x_k$ is not close enough to the local minimizer (maximizer). In our test the stopping criterion always guaranteed the required accuracy by choosing the parameters $\delta_x$ and $\delta_f$ appropriately.

*The criterion:*
If both the set of inequalities

$$\left|x_{k+1}^i - x_k^i\right| \leq \delta_x \left|x_{k+1}^i\right|, \quad i = 1, \ldots, n, \tag{3}$$

and the inequality

$$|f(x_{k+1}) - f(x_k)| \leq \delta_f |f(x_{k+1})|, \tag{4}$$

are fulfilled, terminate the algorithm.

Here $\delta_x$ and $\delta_f$ are the required relative errors for the argument and function value at the solution, respectively.

## 4. Constrained minimization

The new implementation of Shor's *r*-algorithm concerns also the solution of constrained minimization problems by taking into account the constraints

$$h(x) \leqq 0, \tag{5}$$

where $h(x): \mathbb{R}^n \to \mathbb{R}^m$ is a vector valued proper function, by the method of exact penalization. Here we assume that the first $m_e$ constraints are equalities and the next $m_n$, $m = m_e + m_n$, are inequalities.

Minimization of the function $f(x)$ subject to the constraints (5) can be reformulated in its equivalent form:
*minimize*

$$F(x) = f(x) + \lambda \max\Big(0, \max_{i=1,\ldots,m_e} |h_i(x)|, \max_{j=m_e+1,\ldots,m} h_j(x)\Big), \tag{6}$$

where $\lambda$ is a penalty coefficient satisfying the inequality $\lambda > \|u^\star\|_\infty$, $u^\star$ is a vector of the optimal dual parameters corresponding to the constraints (5).

Since no a priori information is available on the exact value $u^\star$, the penalty coefficient has to be chosen sufficiently large (but finite) with the aim to satisfy the constraints. Even though, it may happen that the rate of decrease of the objective function at an infeasible point is still higher than $\lambda$. This may lead to an infinite solution of the minimization problem (6) and hence, to no solution of the initial convex programming problem. On the other hand, the use of an extremely large value $\lambda$, which makes the penalty function extremely steep at the boundary of a feasible set, may prevent to obtaining the exact solution of the problem.

The above mentioned reasons perceive the use of self-adjusting penalties that would make it possible to solve the problem at one run of a minimization algorithm. An adjusting procedure should provide desirably the use of the smallest admissible value $\lambda$ at a solution point. This requires possibly both, reduction and increase of a penalty coefficient rather often and the need to restart anew every time $\lambda$ takes a new value, meaning the need of resetting the transformation matrix $B$ and hence, a loss of the collected data.

The aforesaid was fully taken into account when designing the penalty adjusting procedure as follows.

It starts with $\lambda = 1$. At each iteration $k$, where the two successive points $x_{k-1}$ and $x_k$ are infeasible by means of the constraints (5), we calculate

$$\tilde{\lambda} = -q_1 \frac{f(x_k) - f(x_{k-1})}{\|x_k - x_{k-1}\|}. \tag{7}$$

Next, we compare the two values $\lambda$ and $\tilde{\lambda}$ and if $\tilde{\lambda} > q_2\lambda$, we set $\lambda = \tilde{\lambda}$.

The choice of the constants $q_1$ and $q_2$ should be based on a practical experience and the only condition $q_1 > q_2 > 1$. In particular, the values $q_1 = 15$ and $q_2 = 1.2$ provide a satisfactory performance of the algorithm. The latter means that the constraints are not over penalized and $\lambda$ takes a new, larger, value, if “it seems” to be necessary. By doing this, one guarantees obtaining a finite solution for the problem (6) in practice.

On the other hand, a better efficiency might be achieved by a reduction of the value $\lambda$ when it becomes reasonable. Assume for a number $K$ of successive iterations, the inequality

$$\|g_F(x_k)\| \le \lambda/r_1, \quad g_F(x_k) \in \partial F(x_k), \tag{8}$$

has been taking place. Here

$$k \in \{k : \exists i(k) \leq m_e : h_{i(k)}(x_k) \neq 0 \text{ or } \exists j(k) > m_e : h_{j(k)}(x_k) > 0\}$$

and $r_1 > 1$ is a constant. The fulfillment of inequality (8) for at least $K = 20$ successive infeasible points $x_k$ provides a strong basis for a safe reduction of $\lambda$ in $r_2 < r_1$ times. Practically, the values $r_1 = 100$ and $r_2 = 10$ are used.

## 5. The program

The program `solvopt` is concerned with minimization or maximization of nonlinear, possibly non-smooth objective functions and with the solution of nonlinear programming problems taking into account constraints by the method of exact penalization. Recently, the routine is available as an M-function to be used with Matlab, as a Fortran (MS Fortran, HP-UX Fortran 90) subroutine and as a C (MS VC, HP-UX C) function. We provide also a set of standard test functions for local optimization algorithms and an extensive manual.

The routine `solvopt` takes very few arguments at a call. These are a starting point, entry names of routines, which calculate the objective function value and the gradient at a point, a vector of optional parameters and the names of routines, which calculate the maximal residual for a set of constraints and the gradient of a function $h_j(x)$ taking the maximal positive value at a point $x$. The optional parameters and their default values are discussed in the manual for the program [3]. Here, a few of them are to be mentioned. It is assumed that the required accuracy for the argument $\delta_x = 10^{-4}$ (see (3)) and for the function $\delta_f = 10^{-6}$ (see (4)), the coefficient for space dilation $\alpha = 2.5$, the factor for the first trial step size $c = 1$ (see (1)) and an adjusting procedure for $\gamma$ is invoked (see (2)).

### 5.1. *Gradient approximation*

Calculating gradients, when they are not user-supplied, is a very important issue in view of the efficiency and robustness of a minimization algorithm. In *SolvOpt* package, the routine `apprgrdn` performs the finite difference approximation of the gradient. Normally, it calculates the forward differences with the exception of the case, when the coordinates of a point are smaller than 1 in absolute values. If the latter takes place, the routine calculates the central differences for the coordinates that have small absolute values.

The stepsizes which are used when calculating forward (central) differences are also of a great importance. This specially concerns the case, when the objective function cannot be calculated precisely, i.e. it is perturbed by a noise (see Table 6). The user has to estimate the upper bound of a noise and choose an appropriate relative stepsize for the argument, since this cannot be done automatically.

### 5.2. *Performance of the algorithm*

When *SolvOpt* was developed, the goal was to design an algorithm for smooth and non-smooth optimization problems, which combines efficiency, robustness and accuracy. Many

*Table 1.* The solution for UNC-problems (the tests by Moré et al.) by *SolvOpt* with user-supplied analytically determined gradients and standard starting points.

| Function name (dimension) | Known minimum | Function value at the solution | Rel. error for the minimizer | $N_f$ | $N_g$ |
|---|---|---|---|---|---|
| Rosenbrock (2) | 0.00000e+00 | 6.51487e-14 | 4.57175e-07 | 177 | 38 |
| Freudenstein and Roth (2) | 4.89843e+01 | 4.89843e+01 | 2.19319e-04 | 92 | 27 |
| Powell Badly Scaled (2) | 0.00000e+00 | 4.15387e-13 | 5.77221e-04 | 754 | 149 |
| Brown Badly Scaled (2) | 0.00000e+00 | 2.81555e-15 | 4.66825e-14 | 222 | 37 |
| Beale (2) | 0.00000e+00 | 9.73165e-16 | 3.29049e-08 | 94 | 20 |
| Jennrich and Sampson (2) | 1.24362e+02 | 1.24362e+02 | 1.88848e-05 | 56 | 20 |
| Helical Valley (3) | 0.00000e+00 | 2.67613e-14 | 4.12264e-08 | 153 | 36 |
| Bard (3) | 8.21487e-03 | 8.21488e-03 | 2.95678e-05 | 107 | 33 |
| Gaussian (3) | 1.12793e-08 | 1.12793e-08 | 1.90770e-05 | 71 | 19 |
| Meyer (3) | 8.79458e+01 | 8.79459e+01 | 1.70764e-05 | 2246 | 521 |
| Gulf Research and Dvlp. (3) | 0.00000e+00 | 2.97735e-15 | 1.09601e-05 | 1152 | 258 |
| Box 3-Dimensional (3) | 0.00000e+00 | 4.73124e-13 | 1.32117e-06 | 116 | 23 |
| Powell Singular (4) | 0.00000e+00 | 1.42035e-13 | 2.52338e-04 | 133 | 31 |
| Wood (4) | 0.00000e+00 | 7.05286e-15 | 6.70780e-08 | 248 | 56 |
| Kowalik and Osborne (4) | 3.07506e-04 | 3.07506e-04 | 6.01813e-05 | 108 | 36 |
| Brown and Dennis (4) | 8.58222e+04 | 8.58222e+04 | 2.59210e-05 | 158 | 53 |
| Osborne 1 (5) | 5.46489e-05 | 5.46489e-05 | 1.94239e-04 | 305 | 89 |
| Biggs EXP6 (6) | 5.65565e-03 | 5.65565e-03 | 9.59768e-06 | 141 | 43 |
| Osborne 2 (11) | 4.01377e-02 | 4.01377e-02 | 1.73261e-05 | 216 | 75 |
| Watson (9) | 1.39976e-06 | 1.39976e-06 | 1.81276e-05 | 304 | 97 |
| Extended Rosenbrock (10) | 0.00000e+00 | 1.78220e-13 | 4.75103e-07 | 321 | 81 |
| Extended Powell Singular (4) | 0.00000e+00 | 1.42035e-13 | 2.52338e-04 | 133 | 31 |
| Penalty I (4) | 2.24998e-05 | 2.24998e-05 | 5.60111e-05 | 254 | 67 |
| Penalty II (4) | 9.37629e-06 | 9.37629e-06 | 8.78873e-05 | 986 | 264 |
| Variably Dimensioned (10) | 0.00000e+00 | 2.08035e-13 | 2.51484e-07 | 399 | 86 |
| Trigonometric (10) | 2.79506e-05 | 2.79506e-05 | 0.00000e+00 | 143 | 49 |
| Discrete Boundary Value (10) | 0.00000e+00 | 8.52721e-14 | 4.58385e-06 | 161 | 40 |
| Discrete Integral Equat. (10) | 0.00000e+00 | 6.85021e-13 | 7.15789e-06 | 111 | 27 |
| Broyden Tridiagonal (10) | 0.00000e+00 | 8.91327e-14 | 7.27062e-07 | 119 | 30 |
| Broyden Banded (10) | 0.00000e+00 | 6.75979e-13 | 9.80184e-07 | 141 | 30 |
| Linear—Full Rank (10) | 1.00000e+01 | 1.00000e+01 | 3.23741e-05 | 379 | 113 |
| Linear—Rank 1 (10) | 4.63415e+00 | 4.63415e+00 | 0.00000e+00 | 133 | 24 |
| Linear—Rank 1 with Zero Columns and Rows (10) | 6.13514e+00 | 6.13514e+00 | 0.00000e+00 | 289 | 24 |

*Table 2.* The solution for UNC-problems (the tests by Moré et al.) by *SolvOpt* without user-supplied gradients and standard starting points.

| Function name (dimension) | Known minimum | Function value at the solution | Rel. error for the minimizer | $N_f$ |
|---|---|---|---|---|
| Rosenbrock (2) | 0.00000e+00 | 7.50463e-15 | 1.67244e-07 | 285 |
| Freudenstein and Roth (2) | 4.89843e+01 | 4.89843e+01 | 2.40831e-04 | 147 |
| Powell Badly Scaled (2) | 0.00000e+00 | 4.70720e-21 | 1.97603e-08 | 1241 |
| Brown Badly Scaled (2) | 0.00000e+00 | 6.14504e-13 | 7.74628e-13 | 293 |
| Beale (2) | 0.00000e+00 | 1.78959e-14 | 1.20877e-07 | 113 |
| Jennrich and Sampson (2) | 1.24362e+02 | 1.24362e+02 | 7.09685e-06 | 90 |
| Helical Valley (3) | 0.00000e+00 | 2.11807e-15 | 4.11560e-09 | 283 |
| Bard (3) | 8.21487e-03 | 8.21488e-03 | 1.11801e-05 | 189 |
| Gaussian (3) | 1.12793e-08 | 1.12793e-08 | 1.90422e-05 | 118 |
| Meyer (3) | 8.79458e+01 | 8.79459e+01 | 7.12908e-05 | 4584 |
| Gulf Research and Dvlp. (3) | 0.00000e+00 | 7.63487e-16 | 2.55022e-05 | 2119 |
| Box 3-Dimensional (3) | 0.00000e+00 | 3.31440e-15 | 2.53636e-07 | 165 |
| Powell Singular (4) | 0.00000e+00 | 1.41568e-13 | 2.39728e-04 | 269 |
| Wood (4) | 0.00000e+00 | 1.62095e-13 | 3.66083e-07 | 486 |
| Kowalik and Osborne (4) | 3.07506e-04 | 3.07506e-04 | 6.85225e-06 | 253 |
| Brown and Dennis (4) | 8.58222e+04 | 8.58222e+04 | 3.00919e-05 | 243 |
| Osborne 1 (5) | 5.46489e-05 | 5.46489e-05 | 1.99793e-04 | 882 |
| Biggs EXP6 (6) | 0.00000e+00 | 6.92153e-14 | 1.49201e-05 | 1382 |
| Osborne 2 (11) | 4.01377e-02 | 4.01377e-02 | 3.71987e-05 | 1024 |
| Watson (9) | 1.39976e-06 | 1.39977e-06 | 3.43343e-03 | 1055 |
| Extended Rosenbrock (10) | 0.00000e+00 | 1.87397e-13 | 4.10093e-07 | 1968 |
| Extended Powell Singular (4) | 0.00000e+00 | 1.41568e-13 | 2.39728e-04 | 269 |
| Penalty I (4) | 2.24998e-05 | 2.24998e-05 | 3.85047e-05 | 483 |
| Penalty II (4) | 9.37629e-06 | 9.37629e-06 | 2.64325e-05 | 2387 |
| Variably Dimensioned (10) | 0.00000e+00 | 5.29850e-13 | 4.45526e-07 | 333 |
| Trigonometric (10) | 2.79506e-05 | 2.79506e-05 | 0.00000e+00 | 459 |
| Discrete Boundary Value (10) | 0.00000e+00 | 6.59758e-14 | 3.43634e-06 | 586 |
| Discrete Integral Equat. (10) | 0.00000e+00 | 6.28448e-14 | 3.16254e-06 | 349 |
| Broyden Tridiagonal (10) | 0.00000e+00 | 4.26345e-14 | 6.94252e-07 | 420 |
| Broyden Banded (10) | 0.00000e+00 | 4.98694e-13 | 9.26124e-07 | 376 |
| Linear—Full Rank (10) | 1.00000e+01 | 1.00000e+01 | 1.43688e-05 | 305 |
| Linear—Rank 1 (10) | 4.63415e+00 | 4.63415e+00 | 0.00000e+00 | 1180 |
| Linear—Rank 1 with Zero Columns and Rows (10) | 6.13514e+00 | 6.13514e+00 | 0.00000e+00 | 847 |

runs with test functions known to be a challenge for any optimization routine indicate that these design goals have been achieved.

Despite *SolvOpt* is claimed to be an optimization tool for almost every problem, the weak point of the code is seen in its general applicability. The latter means that it cannot compete in efficiency with the optimization codes designed specifically for a particular class of problems.

Below we provide representative samples of our tests. These have been run with the standard starting points, unless stated otherwise. For all runs the default values for the parameters were used.

***5.2.1. Unconstrained minimization of smooth functions.*** Tables 1 and 2 contain the results for known differentiable functions by Moré et al. [5]. The first column of both tables displays the title and the dimension of the test function. The second column provides the known function value at the minimum. The third displays the minimum value obtained by the M-function `solvopt` with default optional parameter settings (see above). The fourth gives the relative error (in terms of $\ell_\infty$-norm) for the minimizer obtained. The numbers of function ($N_f$) and gradient ($N_g$) evaluations are presented in columns 5 and 6 respectively.

***5.2.2. Unconstrained minimization of non-smooth functions.*** The two Tables 3 and 4 contain the results for the minimization of non-smooth functions. As previously, the optional parameters take the default values.

*Table 3.* The solution for the selected non-smooth problems by *SolvOpt* with user-supplied analytically determined subgradients.

| Function name (dimension) | Known minimum | Function value at the solution | Rel. error for the minimizer | $N_f$ | $N_g$ |
|---|---|---|---|---|---|
| Shor's Piece-wise Quadr. (5) | 2.260016e+01 | 2.260016e+01 | 1.02286e-04 | 179 | 61 |
| Exact penalty function for the Shell Dual Problem (15) | 3.234868e+01 | 3.234868e+01 | 1.04073e-03 | 1081 | 311 |
| Exact penalty function for the Ill-Conditioned LPP (15) | -2.004200e+01 | -2.004200e+01 | 1.11044e-04 | 740 | 246 |
| Lemarechal's MaxQuad (10) | -8.414083e-01 | -8.414083e-01 | 1.79634e-04 | 324 | 109 |

*Table 4.* The solution for the selected non-smooth problems by *SolvOpt* without user-supplied subgradients.

| Function name (dimension) | Known minimum | Function value at the solution | Rel. error for the minimizer | $N_f$ |
|---|---|---|---|---|
| Shor's Piece-wise Quadr. (5) [6] | 2.260016e+01 | 2.260016e+01 | 6.90639e-05 | 539 |
| Exact penalty function for the Shell Dual Problem (15) [2] | 3.234868e+01 | 3.234869e+01 | 1.25461e-02 | 5892 |
| Exact penalty function for the Ill-Conditioned LPP (15) [2] | -2.004200e+01 | -2.004200e+01 | 3.14566e-04 | 780 |
| Lemarechal's MaxQuad (10) [4] | -8.414083e-01 | -8.414083e-01 | 9.00188e-04 | 1289 |

One can find many more test results at *SolvOpt* web page. In particular, we have collected there the results for Moré set of tests at a) 10-, 100-, 1000-fold standard starting points and b) $\tilde{f}(\cdot) = 10^{16} f(\cdot)$ and $\tilde{f}(\cdot) = 10^{-8} f(\cdot)$. These sets of tests were run for the both cases, when the gradients are analytically calculated and approximated by the finite differences. By these runs, we have proven that the convergence behavior of *SolvOpt* depends insignificantly on a starting point. This is true for both "flat" and "steep" functions too.

***5.2.3. Constrained minimization.*** The following runs illustrate minimization of a function under constraints. In Table 5, the first column provides the title (identifier) of a test problem. The second one displays the number of variables ($n$). The numbers of equality ($m_e$) and inequality ($m_n$) constraints are found in the third and the fourth columns respectively. The fifth column provides the function value at the solution found and the other four columns display the values for the counters: objective function values ($N_f$), gradients ($N_g$) of the

*Table 5.* The solution for constrained problems selected from the CUTE [1] by *SolvOpt* with user-supplied gradients.

| Problem identifier | $n$ | $m_e$ | $m_n$ | Minimum | $N_f$ | $N_g$ | $N_{Fcn}$ | $N_{Gcn}$ |
|---|---|---|---|---|---|---|---|---|
| ANTEN1 | 7 | 0 | 333 | .854175437D-01 | 634 | 195 | 634 | 123 |
| ANTEN2 | 7 | 0 | 333 | .113104704D+00 | 793 | 245 | 793 | 161 |
| ANTEN3 | 7 | 0 | 333 | .220519858D+00 | 376 | 125 | 376 | 122 |
| ROBOTA | 21 | 0 | 1458 | .105606403D+01 | 1106 | 344 | 1106 | 218 |
| ROBOTB | 21 | 0 | 1458 | .237778642D+01 | 2172 | 676 | 2172 | 442 |
| ROBOTC | 21 | 0 | 1458 | .147365200D+01 | 1853 | 548 | 1853 | 321 |
| DEMBO1B (SCALING) | 12 | 0 | 15 | .316867632D+01 | 902 | 273 | 902 | 80 |
| DEMBO4C | 9 | 0 | 23 | .360509830D+01 | 4281 | 1326 | 4281 | 1325 |
| DEMBO7N16 | 16 | 0 | 51 | .174793463D+01 | 2449 | 787 | 2449 | 731 |
| HS114EXTD | 18 | 11 | 28 | -.176879039D+04 | 1667 | 540 | 1667 | 537 |
| HS118 | 15 | 0 | 59 | .664826996D+03 | 1584 | 516 | 1584 | 361 |
| HS118M1 | 15 | 0 | 59 | .128559996D+05 | 677 | 219 | 677 | 126 |
| HS118M2 | 15 | 1 | 59 | .662121592D+03 | 1188 | 396 | 1188 | 393 |
| HS119 COLVILLE NO 7 | 16 | 8 | 32 | .244899701D+03 | 1236 | 405 | 1236 | 400 |
| HS108XMOD O | 9 | 0 | 14 | -.674974711D+00 | 495 | 153 | 495 | 91 |
| HS69 | 4 | 2 | 8 | -.956711996D+03 | 3537 | 963 | 3537 | 908 |
| HS84ORIG | 5 | 0 | 16 | -.528033302D+07 | 501 | 114 | 501 | 60 |
| HS84SCAL | 5 | 0 | 16 | -.528033517D+02 | 646 | 161 | 646 | 82 |
| HS85ORIG | 5 | 0 | 48 | -.190515360D+01 | 315 | 86 | 315 | 58 |
| HS85REG | 5 | 0 | 48 | -.190515360D+01 | 315 | 86 | 315 | 58 |
| HIM13BOXPROBLEM | 12 | 7 | 16 | -.523801553D+07 | 22061 | 6476 | 22061 | 6471 |
| HIM20PAVIANIN24 | 24 | 14 | 32 | .556580411D+01 | 3943 | 1250 | 3943 | 1145 |
| HIM6MINOS | 45 | 16 | 45 | .122502412D+06 | 151 | 27 | 151 | 26 |
| HIM6SPELLUCCI | 45 | 16 | 45 | -.191037195D+04 | 3939 | 1309 | 3939 | 1266 |

*Table 6.* Minimization of Rosenbrock's function in presence of an additive noise.

| No. | Function value at solution $\bar{x}$ | $\ell_\infty$-norm for $\bar{x} - x_\star$ | $N_f$ | No. | Function value at solution $\bar{x}$ | $\ell_\infty$-norm for $\bar{x} - x_\star$ | $N_f$ |
|---|---|---|---|---|---|---|---|
| 1 | 8.04800e-10 | 4.27244e-05 | 413 | 2 | 9.61321e-12 | 6.19792e-06 | 244 |
| 3 | 1.83812e-09 | 8.50340e-05 | 225 | 4 | 1.10704e-09 | 4.80238e-05 | 283 |
| 5 | 2.51601e-09 | 5.12358e-05 | 242 | 6 | 6.00402e-09 | 1.50768e-04 | 423 |
| 7 | 3.02638e-11 | 1.02842e-05 | 275 | 8 | 2.86855e-09 | 1.53632e-05 | 234 |
| 9 | 5.76961e-10 | 4.46547e-05 | 286 | 10 | 6.46889e-10 | 4.44173e-05 | 274 |
| 11 | 3.03039e-09 | 9.66342e-05 | 279 | 12 | 3.20924e-09 | 1.00841e-04 | 264 |
| 13 | 4.26250e-12 | 3.01130e-06 | 262 | 14 | 5.87820e-11 | 5.37228e-06 | 237 |
| 15 | 4.56702e-09 | 1.19603e-04 | 258 | 16 | 5.95747e-09 | 1.29142e-04 | 245 |
| 17 | 2.12739e-12 | 2.88789e-06 | 311 | 18 | 4.04631e-09 | 1.27245e-04 | 254 |
| 19 | 1.15334e-09 | 5.53757e-05 | 281 | 20 | 4.22626e-10 | 1.39456e-05 | 267 |
| 21 | 9.60271e-09 | 1.96189e-04 | 282 | 22 | 3.47445e-10 | 1.10488e-06 | 258 |
| 23 | 4.60912e-11 | 9.56332e-06 | 278 | 24 | 1.04007e-10 | 1.66566e-05 | 276 |
| 25 | 6.04415e-09 | 1.36153e-04 | 245 | 26 | 4.56546e-09 | 6.76059e-05 | 263 |
| 27 | 4.32422e-10 | 4.16216e-05 | 257 | 28 | 1.57247e-10 | 2.41977e-05 | 263 |
| 29 | 2.32686e-09 | 9.65815e-05 | 284 | 30 | 3.48260e-10 | 2.88937e-05 | 233 |

objective function, maximal residuals for constraints ($N_{Fcn}$) and gradients for constraints ($N_{Gcn}$) respectively. These results were obtained by the FORTRAN subroutine `solvopt`.

***5.2.4. Minimization of a perturbed function.*** Another very important feature of any optimization routine is its robustness under bounded perturbations. In many practical applications one faces the necessity of optimizing a perturbed function or a function which cannot be computed exactly, i.e., it can be computed only with finite accuracy only. The table below shows the robustness of *SolvOpt* in this respect by minimizing perturbed Rosenbrock's function. The values of Rosenbrock's function were perturbed by additive noise by adding

```
1.e-7*(rand(1)-0.5)
```

to the function value. The results presented in Table 6 have been obtained at the required relative errors $\delta_x = 10^{-4}$ and $\delta_f = 10^{-5}$. The lower bound for the relative stepsize used in the difference approximation of gradients has been set to $10^{-6}$.

## 6. Concluding remarks

The data on test runs presented above are certainly incomplete. In total, about 20,000 runs have been made with various test functions and at dimensions from 2 up to 500. With all these runs, *SolvOpt* has shown a great robustness in achieving the required accuracy. This proves *SolvOpt* to be a very useful optimization tool. Nevertheless, the authors do not claim the robustness of the code in every single case. Obviously, there is no optimization code that never fails. The case of a particular difficulty for *SolvOpt* is a ravine-like function with very steep and almost parallel walls and a flat bottom. As a rule, this is caused by a finite accuracy of computations.

Concerning constrained minimization, the use of penalty functions restricts the area of applicability of *SolvOpt* to objective functions defined everywhere in $\mathbb{R}^n$. Obviously, one can complete the function definition by introducing a linear penalty function taking a nonzero value outside the domain. However, this requires from the user to define a penalty function and to choose a proper penalty coefficient a priori.

The Matlab, C and Fortran source codes for *SolvOpt* are available for downloading from `http://bedvgm.kfunigraz.ac.at:8001/alex/solvopt/` (the Internet addresses are under changes). One can find also the online manual and its postscript version, the test descriptions and the results at the web site.

Since the newest version 1.1 had been announced in August, 1997, *SolvOpt* has got over 1200 users and this number grows up constantly by about 3 downloads per day in average.

The present paper does not provide a comparison of the efficiency of *SolvOpt* and other existing implementations of the $r$-algorithm. The reason is that there is no source code available in public except *SolvOpt* and we could not find a single comprehensive description of test results that would be useful to compare with.

## Acknowledgments

The authors acknowledge support by FWF (Austria) under grant F003.

## References

1. I. Bongartz, A.R. Conn, N.I.M. Gould, and Ph.L. Toint, “CUTE: constrained and unconstrained testing environment,” Research Report, IBM, T.J. Watson Research Center, Yorktown, USA, 1993.
2. K.C. Kiwiel, Methods of Descent for Nondifferentiable Optimization, Springer-Verlag: Berlin, 1985. Lecture Notes in Mathematics, vol. 1133.
3. A.V. Kuntsevich and F. Kappel, The solver for local nonlinear optimization problems (version 1.1), University of Graz, Graz, 1997.
4. C. Lemarechal and R. Mifflin, (Eds.), Nonsmooth Optimization, Pergamon Press: Oxford, 1978.
5. J.J. Moré, B.S. Garbow, and K.E. Hillstrom, “Testing unconstrained optimization software,” ACM Transactions on Mathematical Software, vol. 7, no. 1, pp. 17–41, 1980.
6. N.Z. Shor. Minimization methods for Non-Differentiable Functions, Springer-Verlag: Berlin, 1985. Springer Series in Computational Mathematics, vol. 3.