Comput Optim Appl
DOI 10.1007/s10589-008-9196-3

Portfolio optimization by minimizing conditional value-at-risk via nondifferentiable optimization

Churlzu Lim · Hanif D. Sherali · Stan Uryasev

Received: 23 January 2007 / Revised: 31 August 2007

Abstract Conditional Value-at-Risk (CVaR) is a portfolio evaluation function having appealing features such as sub-additivity and convexity. Although the CVaR function is nondifferentiable, scenario-based CVaR minimization problems can be reformulated as linear programs (LPs) that afford solutions via widely-used commercial softwares. However, finding solutions through LP formulations for problems having many financial instruments and a large number of price scenarios can be time-consuming as the dimension of the problem greatly increases. In this paper, we propose a two-phase approach that is suitable for solving CVaR minimization problems having a large number of price scenarios. In the first phase, conventional differentiable optimization techniques are used while circumventing nondifferentiable points, and in the second phase, we employ a theoretically convergent, variable target value nondifferentiable optimization technique. The resultant two-phase procedure guarantees infinite convergence to optimality. As an optional third phase, we additionally perform a switchover to a simplex solver starting with a crash basis obtained from the second phase when finite convergence to an exact optimum is desired. This three phase procedure substantially reduces the effort required in comparison with the direct use of a commercial stand-alone simplex solver (CPLEX 9.0). Moreover,

C. Lim (✉)
Systems Engineering & Engineering Management, University of North Carolina at Charlotte, Charlotte, NC 28223, USA
e-mail: clim2@uncc.edu

H.D. Sherali
Grado Department of Industrial and Systems Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0118, USA
e-mail: hanifs@vt.edu

S. Uryasev
Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611-6595, USA
e-mail: uryasev@ufl.edu

the two-phase method provides highly-accurate near-optimal solutions with a significantly improved performance over the interior point barrier implementation of CPLEX 9.0 as well, especially when the number of scenarios is large. We also provide some benchmarking results on using an alternative popular proximal bundle nondifferentiable optimization technique.

Keywords Portfolio optimization · CVaR · Nondifferentiable optimization

1 Introduction

1.1 Problem description

In this paper, we are concerned with finding an optimal portfolio that is comprised of a number of financial instruments under price uncertainty. Although a commonly used metric of portfolio optimization can be either expected return or risk, we are particularly interested in minimizing a risk function. Let $\mathbf{x} \in R^n$ denote a portfolio vector indicating the fraction of investment of some available budget in each of n financial instruments. (All vectors are assumed to be column vectors, unless otherwise noted.) Furthermore, let $F(\mathbf{x})$ be a function measuring the risk of portfolio $\mathbf{x}$, given some probabilistic price (or return) distribution. Assuming that no short positions are allowed, an optimal portfolio can be determined by solving the following optimization problem.

$$\text{Minimize} \quad F(\mathbf{x}) \tag{1a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_i = 1 \tag{1b}$$

$$\mathbf{x} \geq \mathbf{0}. \tag{1c}$$

In addition to Constraints (1b)–(1c), a lower bound on the expected return generated by a portfolio can be enforced in practice in order to accommodate the decision maker's risk-taking aspect. This extension will be discussed and implemented later in this paper.

Among various risk criteria, *Value-at-Risk* (VaR) is a popular measurement of risk representing the percentile of the loss distribution with a specified confidence level. Let $\alpha \in (0, 1)$ and $f(\mathbf{x}, \mathbf{y})$ respectively denote a confidence level and a loss function associated with a portfolio $\mathbf{x}$ and an instrument price (or return) vector $\mathbf{y} \in R^n$. ($f(\mathbf{x}, \mathbf{y}) < 0$ means a positive return.) Then, the VaR function, $\zeta(\mathbf{x}, \alpha)$, is given by the smallest number satisfying $\Phi(\mathbf{x}, \zeta(\mathbf{x}, \alpha)) = \alpha$, where $\Phi(\mathbf{x}, \zeta)$ is the probability that the loss $f(\mathbf{x}, \mathbf{y})$ does not exceed a threshold value ζ (i.e., $\Phi(\mathbf{x}, \zeta) = \Pr[f(\mathbf{x}, \mathbf{y}) \leq \zeta]$). In other words, for any portfolio $\mathbf{x} \in R^n$ and a confidence level α, VaR is the value of ζ such that the probability of the loss not exceeding ζ is α (if ζ is not unique, the smallest such value is taken). However, VaR lacks sub-additivity. Furthermore, when analyzed with scenarios, VaR is nonconvex as well as nondifferentiable, and hence, it is difficult to find a global minimum via conventional optimization techniques. Alternatively, *Conditional VaR* (CVaR), introduced by Rockafellar and Uryasev [24] and further developed in [25], possesses more appealing features such as sub-additivity and convexity, and moreover, it is a coherent risk measure in the sense of Artzner et al. [2].

For continuous distributions, CVaR, also known as the *Mean Excess Loss*, *Mean Shortfall*, and *Tail VaR*, is the conditional expected loss given that the loss exceeds VaR. That is, CVaR is given by (see [24])

$$\phi_\alpha(\mathbf{x}) = (1-\alpha)^{-1} \int_{f(\mathbf{x},\mathbf{y})>\zeta(\mathbf{x},\alpha)} f(\mathbf{x},\mathbf{y})p(\mathbf{y})d\mathbf{y}, \tag{2}$$

where $p(\mathbf{y})$ is a probability density function of $\mathbf{y}$. For general distributions, including discrete distributions, CVaR is a weighted average of VaR and the conditional expectation given by (2) (see [25]). To avoid complications caused by an implicitly defined function $\zeta(\mathbf{x},\alpha)$, Uryasev and Rockafellar [24] have provided an alternative function given by

$$F_\alpha(\mathbf{x},\zeta) = \zeta + (1-\alpha)^{-1} \int_{f(\mathbf{x},\mathbf{y})>\zeta} [f(\mathbf{x},\mathbf{y})-\zeta]p(\mathbf{y})d\mathbf{y}, \tag{3}$$

for which they show that minimizing $F_\alpha(\mathbf{x},\zeta)$ with respect to $(\mathbf{x},\zeta)$ yields the minimum CVaR and its solution. (This statement is again true for general distributions.)

In case that the probability distribution of $\mathbf{y}$ is not available or an analytical solution is difficult, we can exploit price scenarios, which can be obtained from past price data and/or through Monte Carlo simulation. Assume that this price data is equally likely (e.g., random sampling from a joint price distribution). Given price data $\mathbf{y}^j$ for $j=1,\dots,J$, $F_\alpha(\mathbf{x},\zeta)$ can be approximated by

$$\widetilde{F}_\alpha(\mathbf{x},\zeta) = \zeta + [(1-\alpha)J]^{-1} \sum_{j=1}^{J} \max\{f^j(\mathbf{x})-\zeta,\ 0\}, \tag{4}$$

where $f^j(\mathbf{x}) \equiv f(\mathbf{x},\mathbf{y}^j)$. It is straightforward to show that $\widetilde{F}_\alpha(\mathbf{x},\zeta)$ is a convex function when $f^j(\mathbf{x})$ is convex. Assuming differentiability of $f^j(\mathbf{x})$, let $\nabla f^j(\mathbf{x})$ denote the gradient of f^j at $\mathbf{x}$. Then, both $(\nabla f^j(\mathbf{x})^T, -1)^T$ and $\mathbf{0}$ are subgradients of $\max\{f^j(\mathbf{x})-\zeta,\ 0\}$ when $f^j(\mathbf{x})-\zeta=0$, where the superscript T denotes the transpose operator. Therefore, noting that $(\nabla f^j(\mathbf{x})^T, -1)^T \neq \mathbf{0}$, the subdifferential is not a singleton at $(\mathbf{x},\zeta)$ such that $f^j(\mathbf{x})-\zeta=0$, and hence, $\widetilde{F}_\alpha(\mathbf{x},\zeta)$ is nondifferentiable. We can find a portfolio that minimizes CVaR by considering the following nondifferentiable optimization (NDO) problem, which we intend to solve in this paper.

$$\textbf{PF}: \quad \text{Minimize} \quad \zeta + v\sum_{j=1}^{J} \max\{f^j(\mathbf{x})-\zeta,\ 0\} \tag{5a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_i = 1 \tag{5b}$$

$$\mathbf{x} \geq \mathbf{0}, \quad \zeta \text{ unrestricted}, \tag{5c}$$

where $v \equiv [(1-\alpha)J]^{-1}$.

1.2 Literature review

One way to solve PF when the loss function $f^j(\mathbf{x})$ is linear is by using linear programming (LP) as afforded by [24, 35]. Let $\mathbf{p} \in R^n$ denote the vector of purchase prices. Then, the linear loss function is given by $f^j(\mathbf{x}) = (\mathbf{p} - \mathbf{y}^j)^T\mathbf{x}$. Introducing an auxiliary variable vector $\mathbf{z} \in R^J$, PF can be formulated as the following LP problem.

$$\text{Minimize} \quad \zeta + v\sum_{j=1}^{J} z_j \tag{6a}$$

$$\text{subject to} \quad z_j \geq (\mathbf{p} - \mathbf{y}^j)^T\mathbf{x} - \zeta, \quad \forall j = 1, \ldots, J \tag{6b}$$

$$\sum_{i=1}^{n} x_i = 1 \tag{6c}$$

$$\mathbf{x} \geq \mathbf{0}, \quad \mathbf{z} \geq \mathbf{0}, \quad \zeta \text{ unrestricted.} \tag{6d}$$

Rockafellar and Uryasev [24] and Andersson et al. [1] report computational studies using this LP formulation. Due to this simple LP approach, and the aforementioned properties such as convexity and sub-additivity, CVaR has rapidly gained in popularity. Some recent applications include a medium-term integrated risk-management problem for a hydrothermal generation company [12], a short-term bidding strategy in power markets [11], and a trading strategy for the complete liquidation of a financial security under CVaR-based risk constraints [8].

Note that when $f^j(\mathbf{x})$ is nonlinear in $\mathbf{x}$, the corresponding reformulation in (6) becomes a nonlinear program, which may be difficult to solve. Furthermore, since the problem depends on stochastic data, it is desirable to include as many scenarios as possible so that the problem reliably reflects the underlying distribution of prices. However, even for linear loss functions, the size of the LP formulation in (6) drastically increases when a large number of scenarios are used, thereby burdening its solution. This may be problematic because timely decision-making may be critical in the context of financial investments. On the other hand, because the dimension of PF in (5) remains the same as the number of scenarios increases, a direct solution to PF is an attractive choice in practice. In [24], a variable-metric algorithm of Uryasev [34] designed for nondifferentiable optimization was employed for optimizing a NIKKEI portfolio having 11 instruments and 1,000 scenarios. However, due to matrix storage and computational requirements, the use of such variable-metric algorithms in nondifferentiable optimization is limited to problems having a relatively small number of instruments within the portfolio (see [32, 33] for example). With this motivation, we study NDO techniques that can be suitably employed for optimally solving PF with a relatively large number of instruments as well as many price scenarios.

The remainder of this paper is organized as follows. In Sect. 2, we provide a preliminary analysis that will be utilized in developing the proposed algorithms. In Sect. 3, we propose a two-phase method and an augmented three-phase method for solving PF. In particular, the third phase is prescribed for linear loss functions. We report computational results in Sect. 4 for the proposed algorithm and compare its performance with a commercial solver (CPLEX 9.0), as well as with an alternative popular NDO procedure, namely, the proximal bundle algorithm (see [13] for example). Finally, Sect. 5 closes the paper with a summary and conclusions.

2 Preliminary analysis

This section furnishes some basic results pertaining to the analysis of Problem PF, which we utilize in developing the proposed NDO algorithm in Sect. 3. First, noting that $\widetilde{F}_\alpha(\mathbf{x},\zeta)$ is convex on R^{n+1}, a subgradient $(\boldsymbol{\xi}, g)$ at $(\overline{\mathbf{x}}, \overline{\zeta})$ satisfies

$$\widetilde{F}_\alpha(\mathbf{x},\zeta) \geq \widetilde{F}_\alpha(\overline{\mathbf{x}},\overline{\zeta}) + \boldsymbol{\xi}^T(\mathbf{x}-\overline{\mathbf{x}}) + g(\zeta-\overline{\zeta}), \quad \forall(\mathbf{x},\zeta)\in R^{n+1}, \tag{7}$$

where $\boldsymbol{\xi} \in R^n$ and $g \in R$ (see [5]). Such a subgradient vector can be computed as delineated in the following lemma, which is proved in Appendix A.

Lemma 1 *Given* $(\mathbf{x}^k, \zeta^k)$, *let* J_+^k, J_0^k, *and* J_-^k *be subsets of* $\{1,\dots,J\}$ *that correspond to* $f^j(\mathbf{x}^k)-\zeta^k > 0$, $f^j(\mathbf{x}^k)-\zeta^k = 0$, *and* $f^j(\mathbf{x}^k)-\zeta^k < 0$, *respectively. Suppose that* $f^j(\mathbf{x})$ *is a convex differentiable function for all* $j \in \{1,\dots,J\}$. *Let*

$$\boldsymbol{\xi}^k = v\sum_{j\in J_+^k}\nabla f^j(\mathbf{x}^k) + v\sum_{j\in J_0^k}\lambda^j\nabla f^j(\mathbf{x}^k), \tag{8a}$$

$$g^k = 1 - v|J_+^k| - v\sum_{j\in J_0^k}\lambda^j, \tag{8b}$$

where λ^j *is arbitrarily chosen on the interval* $[0,1]$, $\forall j \in J_0^k$, *and where* $\nabla f^j(\mathbf{x}^k)$ *denotes the gradient vector of* $f^j(\mathbf{x})$ *at* $\mathbf{x}^k$. *Then,* $(\boldsymbol{\xi}^k, g^k)$ *is a subgradient of* $\widetilde{F}_\alpha(\mathbf{x},\zeta)$ *at* $(\mathbf{x}^k,\zeta^k)$.

Note that the subdifferential at $(\mathbf{x}^k,\zeta^k)$ is a singleton if and only if $J_0^k = \emptyset$. (Otherwise, more than one subgradient exists as mentioned in the previous section.) This assertion is stated in the following theorem, which will be used in deriving a perturbed differentiable solution in Sect. 3, and is proved in Appendix B.

Theorem 1 *Consider* $(\mathbf{x}^k,\zeta^k)$ *for which there exists* $\varepsilon > 0$ *such that* $f^j(\mathbf{x}) - \zeta \neq 0$, $\forall j = 1,\dots,J$, *for* $(\mathbf{x},\zeta) \in N_\varepsilon(\mathbf{x}^k,\zeta^k) \equiv \{(\mathbf{x},\zeta) : \|(\mathbf{x}^T,\zeta) - ((\mathbf{x}^k)^T,\zeta^k)\| \leq \varepsilon\}$. *Then,* $\widetilde{F}_\alpha$ *is differentiable at* $(\mathbf{x}^k,\zeta^k)$ *with the gradient* $(\boldsymbol{\xi}^k, g^k)$ *given by* (8).

Note that the problem in (5) is constrained by a polytope $\mathbf{X} \equiv \{\mathbf{x}\in R^n : \mathbf{e}^T\mathbf{x} = 1, \mathbf{x} \geq \mathbf{0}\}$, where $\mathbf{e}$ is a vector of n ones. In order for the sequence of solutions to remain feasible, the NDO procedure iterates as follows: a step is taken from the current solution in the direction of the (deflected) subgradient to reach an intermediate point, $\overline{\mathbf{x}}$, which is then projected onto the polytope $\mathbf{X}$ to yield the next iterate. Let $P_{\mathbf{X}}(\overline{\mathbf{x}})$

denote a projection of $\overline{\mathbf{x}}$ onto $\mathbf{X}$, as defined by $P_{\mathbf{X}}(\overline{\mathbf{x}}) \equiv \operatorname{argmin}\{\|\mathbf{x} - \overline{\mathbf{x}}\| : \mathbf{x} \in \mathbf{X}\}$. Then, $P_{\mathbf{X}}(\overline{\mathbf{x}})$ can be obtained by the following sequential projection and collapsing dimension procedure (see Bitran and Hax [6] or Sherali and Shetty [30]).

PROJECTION

Step 0 Initialize $\overline{\mathbf{x}}^1 = \overline{\mathbf{x}}$ and $I^1 = \{1, \dots, n\}$. Set $k = 1$.

Step 1 Let $\rho^k = \sum_{i \in I^k} \overline{x}_i^k$. For each $i \in I^k$, compute $\overline{x}_i^{k+1} = \overline{x}_i^k + \frac{1-\rho^k}{|I^k|}$. If $\overline{x}_i^{k+1} \geq 0, \forall i \in I^k$, put $\widetilde{x}_i = \overline{x}_i^{k+1}, \forall i \in I^k$, and return $\widetilde{\mathbf{x}}$.

Step 2 Let $J^k = \{i \in I^k : \overline{x}_i^{k+1} \leq 0\}$. Set $\widetilde{x}_i = 0, \forall i \in J^k$. Update $I^{k+1} = I^k - J^k$. Increment $k \leftarrow k + 1$, and return to Step 1.

Before proceeding to the next section, we remark that, in addition to the weighted sum and nonnegativity constraints, other constraints can be added to the constraint set of a portfolio optimization problem in order to incorporate an investor's risk preferences and other institutional regulations. Examples include risk constraints, value constraints, and bounds on position changes (see Krokhmal et al. [16, 17] and Krokhmal and Uryasev [15] for example). The projection procedure would become computationally expensive when such side-constraints are present. However, the focus of the present paper is on designing a quick solution method for the basic problem (1), which is still of practical value and poses a challenge due to the large number of scenarios. Incorporating such additional constraints is beyond the scope of this paper, but is a useful consideration for future research.

3 Proposed algorithmic procedures

This section describes a two-phase procedure, and an augmented three-phase method, for solving Problem PF. In the first phase, we implement certain conventional differentiable optimization techniques in conjunction with a subroutine that finds a perturbed differentiable iterate in the vicinity of a current nondifferentiable point. The objective function is differentiable at the resulting perturbed point, and hence, a unique subgradient (i.e., gradient) guarantees a steepest descent direction [29]. Then, in the second phase, a theoretically convergent nondifferentiable optimization scheme is used to solve the problem starting with the solution obtained in the first phase. Note that an arbitrarily chosen subgradient direction may not yield a feasible descent direction at points where the objective function is nondifferentiable. For this reason, subgradient-type methods for solving nondifferentiable optimization problems do not rely on descent directions, but instead focus on finding suitable step-sizes along (deflected) subgradient directions to approach closer in Euclidean norm to the set of optimal solutions, and hence guarantee that the sequence of iterates generated will ultimately converge to an optimal solution. The step-size is usually based on some estimate of the optimal objective value, referred to as the *target value*, which is suitably updated as the algorithm proceeds in order to induce convergence.

The resultant two-phase procedure (called the **Phase I–II Method**) is infinitely convergent to an optimal solution. Additionally, in order to theoretically guarantee *finite* convergence to an exact optimum, we devise a third phase in which the simplex method is implemented using a crash or an *advanced-start basis* that is constructed using the solution obtained via the first two phases. This augmented algorithm is referred to as the **Three-Phase Method**. We remark here that although the second phase itself theoretically guarantees convergence as a stand-alone nondifferentiable optimization technique, the first phase plays an important role in enhancing the performance of the second phase (see [29]). Hence, the first two phases, implemented with a practical limit on the number of iterations, can be used to attain near-optimality, as confirmed by our numerical results in Sect. 4, and the third phase is optionally included as a point of interest. In what follows, we first describe a variable target value method that will constitute the main routine of the procedure (Phase II), and then discuss the pre-processing (Phase I) routine, and the post-processing (Phase III) routine for potentially enhancing the overall performance of the Phase II NDO method.

One viable approach for solving large-scale NDO problems is a (deflected) subgradient algorithm, owing to its relatively modest storage and computational requirements per iteration in comparison with other NDO methods (see [19, 22, 23, 28, 32] for example). When solving Problem PF using such an algorithm, we generate a sequence of iterates $\{(\mathbf{x}^k, \zeta^k)\}$ according to $(\mathbf{x}^{k+1}, \zeta^{k+1}) = (P_{\mathbf{X}}(\mathbf{x}^k + s^k \mathbf{d}_{\mathbf{x}}^k), \zeta^k + s^k d_{\zeta}^k)$, where $s^k \in R$ is a prescribed step-length and $(\mathbf{d}_{\mathbf{x}}^k, d_{\zeta}^k) \in R^{n+1}$ is a prescribed search direction (for example, a subgradient or a deflected subgradient). Although (asymptotic) convergence to optimality can be achieved in theory when a direction is prescribed as a subgradient, i.e., $(\mathbf{d}_{\mathbf{x}}^k, d_{\zeta}^k) = (\boldsymbol{\xi}^k, g^k)$, and the step-length satisfies $s^k \to 0$, $\sum_k s^k \to \infty$ (see [22] for example), such an algorithm is computationally inefficient, and hence, the heuristic method of Held et al. [10] has been widely implemented instead.

On the other hand, among theoretically convergent NDO methods, the *variable target value method* (VTVM) designed by Sherali et al. [32] has been shown to display a promising performance in several computational studies [19, 32, 33]. In VTVM, step-lengths are computed as

$$s^k = \rho(\widetilde{F}(\mathbf{x}^k, \zeta^k) - w)/\|\mathbf{d}^k\|^2, \tag{9}$$

where $\rho \in (0, 1]$ and $\mathbf{d}^k = [(\mathbf{d}_{\mathbf{x}}^k)^T, d_{\zeta}^k]$, and where w is a target value that satisfies $w < \widetilde{F}(\mathbf{x}^k, \zeta^k)$. Lim and Sherali [19] have further enhanced this method by remedying a crawling phenomenon in certain situations and by employing local searches whenever improvements are achieved. In addition, motivated by the cutting plane strategy implemented by Sherali et al. [32], they introduced a sequence of projections onto a class of certain generalized Polyak-Kelley cutting planes (GPKC) within their iterative scheme. In their computational experiments involving the solution of nondifferentiable Lagrangian dual formulations of LPs, this modified VTVM procedure (abbreviated M-VTVM), yielded the best performance when compared with the method of Held et al. [10], the Volume Algorithm of Barahona and Anbil [3], as well as in comparison with another theoretically convergent NDO algorithm, the so-called level algorithm [7, 9]), and its variants. We therefore employ M-VTVM with the GPKC strategy as our Phase II procedure.

Next, we discuss the proposed Phase I procedure. Recently, Sherali and Lim [29] have proposed two pre-processing routines for NDO algorithms, namely, the *perturbation technique* (PT) and the *barrier Lagrangian reformulation* (BLR) method, which are shown to significantly improve the overall performance of M-VTVM when solving nondifferentiable Lagrangian duals of LPs. These methods generate a sequence of differentiable iterates so that standard differentiable (conjugate-) gradient techniques can be employed. Because BLR is customized for solving LPs with bounded variables, whereas PT is better-suited for the form of Problem PF, we adopt this latter procedure in Phase I of the current paper. More specifically, a specialized version of this routine can be designed to bypass nondifferentiable points along the path toward an optimum for our particular problem as follows.

Suppose that we have an iterate $(\mathbf{x}^k, \zeta^k)$ at which $\widetilde{F}_\alpha$ is nondifferentiable and that $\widetilde{F}_\alpha(\mathbf{x}^k, \zeta^k) < \widetilde{F}_\alpha(\mathbf{x}^{k-1}, \zeta^{k-1})$. Note that since $\widetilde{F}_\alpha$ is nondifferentiable, there exists a $j \in \{1, \ldots, J\}$ such that $f^j(\mathbf{x}^k) - \zeta = 0$. To obviate this nondifferentiability, we want to perturb $(\mathbf{x}^k, \zeta^k)$ to find a new point $(\overline{\mathbf{x}}^k, \overline{\zeta}^k)$ that satisfies

$$f^j(\overline{\mathbf{x}}^k) - \overline{\zeta}^k \neq 0, \quad \forall j \in \{1, \ldots, J\} \text{ and } \overline{\mathbf{x}}^k \in \mathbf{X}. \tag{10}$$

To achieve this goal, let $J_-^k \subset \{1, \ldots, J\}$ be an index set such that $f^j(\mathbf{x}^k) - \zeta^k < 0$ for $j \in J_-^k$. Furthermore, let $\overline{\varepsilon} = \min_{j \in J_-^k}\{|f^j(\mathbf{x}^k) - \zeta^k|\}$. If we put $(\overline{\mathbf{x}}^k, \overline{\zeta}^k) = (\mathbf{x}^k, \zeta^k - \widehat{\varepsilon})$, where $\widehat{\varepsilon} \in (0, \overline{\varepsilon})$, then $(\overline{\mathbf{x}}^k, \overline{\zeta}^k)$ satisfies (10), and hence, $\widetilde{F}_\alpha$ is differentiable at this point. Moreover, we have $\widetilde{F}_\alpha(\overline{\mathbf{x}}^k, \overline{\zeta}^k) = \widetilde{F}_\alpha(\mathbf{x}^k, \zeta^k) + (v(J - |J_-^k|) - 1)\widehat{\varepsilon}$. Therefore, if $v(J - |J_-^k|) - 1 \leq 0$, we have that $\widetilde{F}_\alpha(\overline{\mathbf{x}}^k, \overline{\zeta}^k) \leq \widetilde{F}_\alpha(\mathbf{x}^k, \zeta^k) < \widetilde{F}_\alpha(\mathbf{x}^{k-1}, \zeta^{k-1})$ for any value of $\widehat{\varepsilon} \in (0, \overline{\varepsilon})$. However, if $v(J - |J_-^k|) - 1 > 0$, we can further restrict the value of $\widehat{\varepsilon}$ as

$$\widehat{\varepsilon} < \widetilde{\varepsilon} \equiv \min\left\{\frac{\widetilde{F}_\alpha(\mathbf{x}^{k-1}, \zeta^{k-1}) - \widetilde{F}_\alpha(\mathbf{x}^k, \zeta^k)}{v(J - |J_-^k|) - 1}, \overline{\varepsilon}\right\}. \tag{11}$$

In summary, given a nondifferentiable point $(\mathbf{x}^k, \zeta^k)$, we can find a differentiable point $(\overline{\mathbf{x}}^k, \overline{\zeta}^k)$ in the vicinity, whose objective value is still smaller than that of the previous iterate $(\mathbf{x}^{k-1}, \zeta^{k-1})$ where

$$\begin{aligned} \overline{\mathbf{x}}^k &= \mathbf{x}^k \\ \overline{\zeta}^k &= \begin{cases} \zeta^k - 0.5\widetilde{\varepsilon} & \text{if } v(J - |J_-^k|) - 1 > 0 \\ \zeta^k - 0.5\overline{\varepsilon} & \text{otherwise.} \end{cases} \end{aligned}$$

We refer to this process as **Subroutine PT**.

Finally, we address the proposed Phase III post-processing step. Note that NDO algorithms typically yield a quick near-optimal solution. However, in the case when the loss functions are linear, we could further refine this near-optimal solution to an exact optimal solution. Lim and Sherali [18] have proposed one such refinement procedure using an outer-linearization method within a trust region for solving Lagrangian duals of LPs. In contrast, we adopt a simpler approach here that finds a feasible solution to the LP formulation in (6) from the given NDO solution, and accordingly provides an advanced starting basis to a commercial simplex optimizer, CPLEX, using a crash-basis priority order.

To find such an LP feasible solution, let us rewrite (6) as follows.

$$\text{Minimize} \quad \zeta + v\sum_{j=1}^{J} z_j \tag{12a}$$

$$\text{subject to} \quad z_j - s_j - (\mathbf{p} - \mathbf{y}^j)^T\mathbf{x} + \zeta = 0, \quad \forall j = 1, \dots, J \tag{12b}$$

$$\sum_{i=1}^{n} x_i = 1 \tag{12c}$$

$$\mathbf{x} \geq \mathbf{0}, \quad \mathbf{z} \geq \mathbf{0}, \quad \mathbf{s} \geq \mathbf{0}, \quad \zeta \text{ unrestricted}, \tag{12d}$$

where s_j, $\forall j = 1, \dots, J$ are slack variables. (Note that the equality constraints are linearly independent.) Let $(\overline{\mathbf{x}}, \overline{\zeta})$ be a near-optimal solution provided by an NDO algorithm. Then, we can prescribe the following solution that is feasible to (12), where $\psi_j = (\mathbf{p} - \mathbf{y}^j)^T\overline{\mathbf{x}} - \overline{\zeta}$, $\forall j = 1, \dots, J$:

$$z_j = \overline{z}_j \equiv \begin{cases} \psi_j & \text{if } \psi_j > 0 \\ 0 & \text{otherwise}, \ \forall j = 1, \dots, J \end{cases} \tag{13a}$$

$$s_j = \overline{s}_j \equiv \begin{cases} -\psi_j & \text{if } \psi_j < 0 \\ 0 & \text{otherwise}, \ \forall j = 1, \dots, J \end{cases} \tag{13b}$$

$$\mathbf{x} = \overline{\mathbf{x}} \tag{13c}$$

$$\zeta = \overline{\zeta}. \tag{13d}$$

Note that the objective function of the LP formulation in (12) involves only the ζ- and $\mathbf{z}$-variables. Since we want the objective function value at an advanced basis to be as close as possible to that at the given feasible solution to Problem PF, these variables are ascribed the highest priority when constructing a crash-basis. Similarly, $\mathbf{s}$ is given the next level of priority due to its complementary relationship with $\mathbf{z}$ as in (13a)–(13b). Then, the components of $\mathbf{x}$ are prioritized next, in the order from the greatest to the least. With this motivation, we prescribe an advanced basis for the simplex optimizer of CPLEX, by considering the variables for crashing into the basis in the order of $\{\zeta; z_j : \overline{z}_j > 0; s_j : \overline{s}_j > 0; x_j$ ordered in nonincreasing order of $\overline{x}_j, j = 1, \dots, n\}$.

To summarize, the first phase exploits conventional differentiable optimization techniques in which we perturb a nondifferentiable iterate to a differentiable point (via Subroutine PT stated above) whenever we encounter nondifferentiability. At such a perturbed solution, we apply three differentiable optimization strategies, namely, the steepest descent method (SD), the memoryless BFGS method of [27] (MBFGS), and the quasi-Newton-based deflected gradient scheme of [31] (SU). (The latter two strategies are chosen based on the computational results of [29]. In our implementation, these methods are restarted with the negative gradient direction whenever either $n+1$ iterations have been performed since the last restart, or when the prescribed direction is not a descent direction as detailed in the algorithm described below.) A single quadratic-fit line-search is used along this search direction. After K_I iterations of the first phase, we next implement M-VTVM for K_{II} iterations in the second phase in order to find a near-optimal solution. Finally, as an add-on step for the Three-Phase Method, we prescribe a crash-basis to CPLEX for Problem (12) based on the attained near-optimal solution and determine an exact optimal solution using the simplex algorithm in the third phase. Abbreviating $\mathbf{x}_\zeta^k = ((\mathbf{x}^k)^T, \zeta_k)^T$, $\boldsymbol{\xi}_g^k = ((\boldsymbol{\xi}^k)^T, g^k)^T$, and $\widetilde{F}^k = \widetilde{F}_\alpha(\mathbf{x}_\zeta^k)$, the proposed method can be summarized as follows. (See Appendix C for a detailed algorithmic statement along with specific parameter values.)

Phase I: PT

Step 0 Select an initial iterate $\mathbf{x}_\zeta^1$, at which $\widetilde{F}_\alpha$ is differentiable (call Subroutine PT if necessary). Select termination parameters $\varepsilon > 0$ and $K_I > 0$. Put $k = 1$.

Step 1 Terminate if $\|\boldsymbol{\xi}_g^k\| < \varepsilon$. If $k > K_I$, go to Phase II. Otherwise, proceed to Step 2.

Step 2 Compute the next search direction $\mathbf{d}^k$ depending on the deflected subgradient scheme employed (SU or BFGS).

Step 3 If $\mathbf{d}^k$ is not a descent direction, replace it by the subgradient. Perform a single quadratic-fit line-search along the direction $\mathbf{d}^k$ (and project onto $\mathbf{X}$ using the PROJECTION scheme of Sect. 2, if necessary) to obtain the next iterate $\mathbf{x}_\zeta^{k+1}$. If $\widetilde{F}^{k+1}$ is nondifferentiable at $\mathbf{x}_\zeta^{k+1}$, call Subroutine PT. Increment $k \leftarrow k+1$ and return to Step 1.

Phase II: M-VTVM

Step 0 Select algorithmic parameters $\varepsilon_0 > 0$, $K_{II} > 0$, the initial target value w, and the initial iterate $\mathbf{x}_\zeta^1 = \mathbf{x}_\zeta^{K_I}$. Put $k = 1$.

Step 1 Call Subroutine GPKC to compute $\mathbf{x}_\zeta^{k+1}$ (replace $\mathbf{x}_\zeta^{k+1} \leftarrow P_{\mathbf{X}}(\mathbf{x}_\zeta^{k+1})$ if $\mathbf{x}_\zeta^{k+1} \notin \mathbf{X}$). Increment $k \leftarrow k+1$. If $\|\boldsymbol{\xi}_g^k\| < \varepsilon_0$ or $k > K_{II}$, terminate the algorithm with the best solution.

Step 2 If sufficient improvement has been achieved, decrease the target value w. If improvement is unsatisfactory, increase w and reset $\mathbf{x}_\zeta^k$ to the best available solution. Return to Step 1.

Subroutine GPKC:

Step i: Select $\delta > 0$, let $\widehat{F} = \widetilde{F}^k - \rho(\widetilde{F}^k - w)$, and define $S_i = \{\mathbf{x} : (\mathbf{x} - \mathbf{x}_\zeta^{k-1})^T \boldsymbol{\xi}_g^{k-i} \geq \widehat{F} - \widetilde{F}^{k-i}\}$ for $i = 0, \ldots, \delta$. If the iterate was just reset to the best available solution, find $\mathbf{x}_\zeta^{k+1} = P_{S_0}(\mathbf{x}_\zeta^k)$ and return to M-VTVM. Otherwise, find $\mathbf{x}_\zeta^{k+1} = P_{S_0 \cap S_1}(\mathbf{x}_\zeta^k)$.

Step ii: If $\delta = 1$, exit the subroutine. Otherwise, put $i = 2$ and proceed to Step iii.

Step iii: If $P_{S_i}(\mathbf{x}_\zeta^{k+1}) \in S_0 \cap S_1$, replace $\mathbf{x}_\zeta^{k+1} \leftarrow P_{S_i}(\mathbf{x}_\zeta^{k+1})$.

Step iv: If $i = \delta$, exit the subroutine. Else, increment $i \leftarrow i+1$, and return to Step iii.

Phase III: SIMPLEX (optional)

Step 1 Given $\bar{\mathbf{x}}_\zeta$, find a feasible solution $\theta \equiv (\bar{\mathbf{z}}, \bar{\mathbf{s}}, \bar{\mathbf{x}}, \bar{\zeta})$ as in (13).

Step 2 Crash variables into the basis in the order of $\{\zeta; z_j : \overline{z}_j > 0; s_j : \overline{s}_j > 0; x_j$ ordered in nonincreasing order of $\overline{x}_j$, $j = 1, \ldots, n\}$. Run the simplex optimizer of CPLEX from this advanced basis specification.

Before proceeding to the next section, it is important to emphasize that Phase III is required in order to guarantee an exact optimal solution in finite time. In practice, however, employing only the Phase I–II Method can serve as an efficient solution procedure that provides a near-optimal solution in a timely manner, which is critical to profitable short-term decision-making such as intra-day trading via online trading applications. Note that the Phase I–II procedure is infinitely convergent to an optimal solution as formalized in the result stated below. Due to this asymptotic convergence behavior, such NDO procedures are typically run for a limited number of iterations (reflected by the foregoing parameters, K_I and K_{II}) in practical implementations (see [29, 32]).

Theorem 2 *Consider the procedure that implements Phase I and II with $K_{II} = \infty$ and $\varepsilon_0 = 0$. Let $F^\star$ denote the optimal objective value. Suppose that the procedure generates an infinite sequence $\{(\mathbf{x}^k, \zeta^k)\}$. (Otherwise, $F^\star$ is at hand finitely.) Then, $\{(\mathbf{x}^k, \zeta^k)\} \to F^\star$.*

Proof See [19]. □

4 Numerical study

In this section, we report numerical results using the proposed procedures and compare their computational performance with a direct implementation of the LP formulation in (6). All algorithmic parameter values are selected as those prescribed in [19, 29, 32, 33] (see Appendix C), where extensive computational studies have revealed favorable and robust performance using these parameter values for a variety of NDO problems. In addition, for the purpose of benchmarking, we ran a variant of the proximal bundle method [13] (see Appendix D), which employs an aggregate subgradient technique that is suitable for solving relatively large-scale NDO problems due to less storage and computing requirements. We implemented an improved version of the proximal bundle algorithm (PBA) as described in Appendix D in place of the first two phases of the Three-Phase Method, applying Phase III as in the proposed method in order to obtain an optimal solution.

We first present computational results using a relatively large number of instruments together with randomly generated scenarios, and next evaluate the solution procedures using two real-world examples obtained from Meucci [20, 21]. All algorithms were coded in C++ and CPLEX 9.0 C++ Concert Technology was used for solving the linear programs. For solving the quadratic subproblem in PBA, we used the CPLEX 9.0 C Callable Library. All runs were implemented on a Dell Power Edge 2600 having dual Pentium-4 3.2 GHz processors and 6 GB of memory.

Table 1 Summary of test instances

Number of scenarios	Number of instruments	Range of optimal objective values
	50	0.287277–0.292212
50,000	100	0.203955–0.207307
	200	0.143248–0.145112

4.1 Results for random test instances

First, we randomly generated test instances from a multivariate normal distribution. The number of instruments were selected as 50, 100, and 200, while the number of scenarios was fixed as 50,000 to provide an adequate approximation to the underlying unknown continuous price distribution. To create a covariance matrix for each size configuration, we generated an $n \times n$ symmetric matrix whose off-diagonal elements were uniformly sampled from the interval [0, 1], and the diagonal elements were then computed to make the matrix diagonally dominant (hence, positive definite). The corresponding mean vector was set as equal to a vector of zeros. Scenarios were generated using the *Triangular Factorization Method* of Scheuer and Stoller [26], which is recommended by Barr and Slezak [4] based on their computational study. For each number of instruments, we created 10 instances, which are summarized in Table 1. The confidence level used in the CVaR function was selected as $\alpha = 0.95$.

Table 2 reports the computational results. All values are averages of the 10 instances having the same number of instruments. In the table, the Three-Phase Method when implemented in conjunction with the SD, SU, and MBFGS strategies in Phase I, is respectively denoted by TPM-SD, TPM-SU, and TPM-MBFGS. The corresponding results for the Phase I–II Method can be gleaned by examining the performance of the TPM procedures at the end of the Phase II. Furthermore, the proximal bundle algorithm implemented in concert with the simplex algorithm is denoted by PBA-SIM, while SIMPLEX and BARRIER respectively stand for the stand-alone simplex algorithm and the interior point barrier function approach implemented within CPLEX 9.0 with a default setting of parameters. GAP-I, GAP-II, and GAP-P represent the percentage optimality gaps in the objective values obtained after Phase I, II, and PBA, respectively, as defined by [(objective value-optimal value)/optimal value]$\cdot$100. Similarly, CPU-I, CPU-II, and CPU-P are the (cumulative) CPU times consumed until the end of Phases I, II, and PBA, respectively. CPU is the overall CPU time for attaining optimality, while ITR denotes the number of iterations required by the simplex optimizer.

First, observe that the proposed Phase I–II Method yields a solution within the optimality gap of $10^{-6}\%$ for all instances with 50 and 100 instruments (reported as zeros). For the 200 instrument cases, the average optimality gap yielded by the Phase I–II Method does not exceed 0.00014% when SD and SU are employed in the first phase, while being less than $10^{-6}\%$ for the MBFGS variant (again reported as zero). As expected, the CPU time consumed by the Phase I–II Method increases almost linearly with the number of instruments (see Fig. 1).

Regarding the Three-Phase Method, all three variants TPM-SD, TPM-SU, and TPM-MBFGS performed comparably. In contrast with the stand-alone simplex al-

Table 2 Computational results

Procedure	Performance measure	Number of instruments		
		50	100	200
TPM-SD	GAP-I (%)	0.00052	0.00175	0.00679
	GAP-II (%)	0	0	0.00014
	CPU-I (s)	17.3	35	75.2
	CPU-II (s)	28.1	56.7	118.8
	CPU (s)	102.9	532.5	2755.3
	ITR	1164.7	4509.7	12592.7
TPM-SU	GAP-I (%)	0.00017	0.00102	0.00340
	GAP-II (%)	0	0	0.00014
	CPU-I (s)	10.7	18.1	45.4
	CPU-II (s)	21.7	40	89.5
	CPU (s)	106.7	555.9	2772.6
	ITR	1289.7	4652.7	12124.1
TPM-MBFGS	GAP-I (%)	0.00062	0.00359	0.02972
	GAP-II (%)	0	0	0
	CPU-I (s)	8.3	17.4	33.7
	CPU-II (s)	19.3	40.9	77.6
	CPU (s)	105.6	544	3041.2
	ITR	1309.8	4484.5	13384.3
PBA-SIM	GAP-P (%)	0.116	0.988	4.317
	CPU-P (s)	2.98	11.13	17.98
	CPU (s)	578.3	5383	14793.1
	ITR	8347.1	28267.7	58222.9
SIMPLEX	CPU (s)	623.683	1561.893	4538.2
	ITR	13976.3	19379.3	28203.6
BARRIER	CPU (s)	195.0	147.6	194.8

gorithm, TPM saves considerable CPU effort. For example, the TPM-SD procedure consumed only 16.5, 34.1, and 60.7% of the CPU times required by the stand-alone simplex algorithm for solving problems having 50, 100 and 200 instruments, respectively. However, the relative superiority of TPM over the stand-alone simplex algorithm declines as the number of instruments increases. This is because the dimension of the nondifferentiable optimization problem in (5) sharply increases (e.g., from 100 to 200), while that of the corresponding LP formulation in (6) does not (e.g., from 50,100 to 50,200). Furthermore, TPM also exhibits improved CPU times over BARRIER for instances having 50 instruments, whereas BARRIER performs better

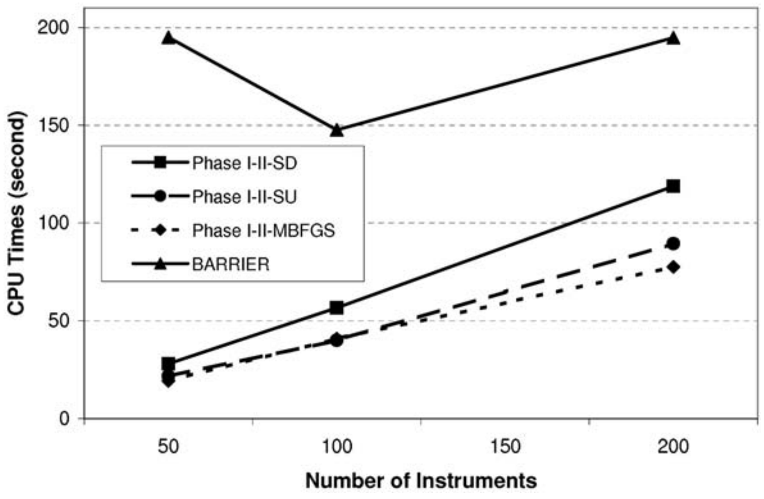

Fig. 1 Illustration of CPU time trajectories for the Phase I–II method and BARRIER

for instances having 100 and 200 instruments. Since the barrier optimizer consumed much smaller CPU times than the stand-alone simplex method did for these two sets of instances, one could recommend the stand-alone barrier function algorithm as the ratio of the number of instruments to that of scenarios increases. However, the Phase I–II Method procedure is itself theoretically convergent to optimality and, as evident from Table 2, found near-optimal solutions within a very small optimality gap (0–0.00014% on average) while consuming only 1.7–4.5% of the computational effort when compared with the stand-alone simplex algorithm, and took only 9.9–61.0% of the CPU time required by BARRIER.

As far as PBA-SIM is concerned, this routine slightly reduced the overall computational effort in comparison with SIMPLEX for the case of 50 instruments, but it significantly worsened in relative performance when the number of instruments was increased. In particular, it actually increased the number of iterations performed by the simplex algorithm. Note that PBA yields fairly large optimality gaps (GAP-P) compared with those of the Phase I–II Method (GAP-II), and hence might not be providing a helpful advanced starting solution to the simplex method. We also observed that PBA quickly meets the stopping criterion in Step 2 even though we used a smaller tolerance value (10^{-10}) than that implemented in [14] (10^{-6}). Because of this premature termination with a poorer quality solution, in order to obtain a better comparison, we measured the CPU times for the Three-Phase Method until it attained the same objective value produced by PBA. In this experiment, all runs terminated in the first phase itself. TPM-SD (TPM-MBFGS) revealed the relatively best (worst) performance with respective average CPU times of 1.12, 3.48, and 12.17 (3.17, 4.77, and 14.4) seconds for the 50, 100, and 200 instrument problems. Hence, Phase I of TPM-SD itself produced the same quality solutions while consuming only 23.0–67.7% of the computational effort required by PBA.

In order to further examine the performance of the Phase I–II Method versus the barrier method, we generated two sets of test instances. The first set was used to study

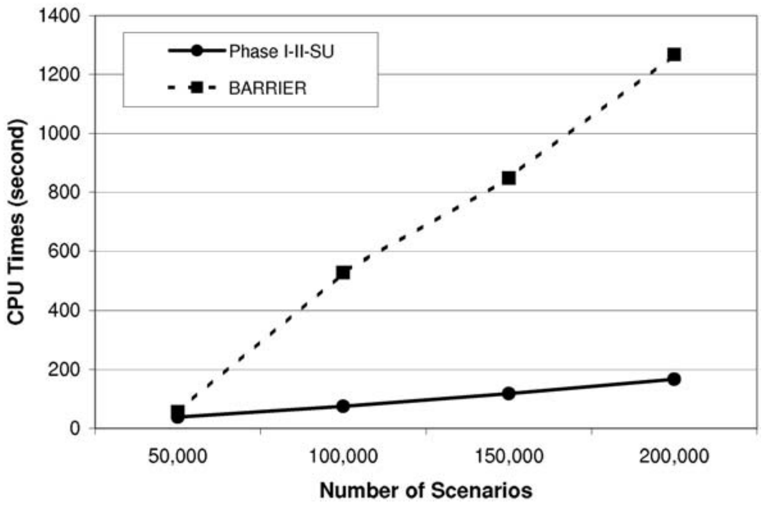

Fig. 2 CPU times for Phase I–II-SU and BARRIER for 100-instrument problems

the trend as the number of scenarios increases, whereas the second set was used to investigate the trend as the number of instruments increases beyond 200. Accordingly, in the first set, we fixed the number of instruments to 100 and randomly generated four instances having 50,000, 100,000, 150,000, and 200,000 scenarios, respectively, and the second set was generated for 250, 500, 750, and 1000 instruments while the number of scenarios was fixed to 50,000.

For the first set, the Phase I–II-SU scheme attained the optimal objective value for all four instances. Figure 2 compares the CPU times of this procedure with those of BARRIER as the number of scenarios increases. Both CPU times have linear trends in the number of scenarios. However, note that the time gap between the two methods drastically increases as the number of scenarios increases. In particular, Phase I–II-SU consumed 167 seconds while BARRIER required 1268 seconds for 200,000 scenarios. The performance trend for the second set is displayed in Fig. 3. Note that BARRIER was unable to solve the problems having 750 and 1,000 instruments due to excessive memory requirements. The Phase I–II-SU procedure yielded the optimal objective value for the instance having 250 instruments while the optimality gap at termination for 500 instruments was $6.7 \times 10^{-4}\%$. The computational effort of Phase I–II-SU seemingly displays a marginally increasing trend as the number of instruments increases. Although the pattern of BARRIER is undemonstrable due to lack of solved instances, BARRIER requires significantly longer CPU times (389 and 1246 seconds) as the number of instruments increases from 250 to 500, as compared with 226 and 504 seconds for Phase I–II-SU. These two experimental results illustrate that the Phase I–II Method can be beneficially employed for problems having a large number of scenarios and/or a large number of instruments.

Finally, we performed another experiment to test the exact TPM procedure versus BARRIER, as also in relation to the Phase I–II Method. For this purpose, we randomly generated 20 instances having 10 instruments, and a variety of scenarios ranging from 50,000 to 1,000,000 in increments of 50,000. The CPU times consumed

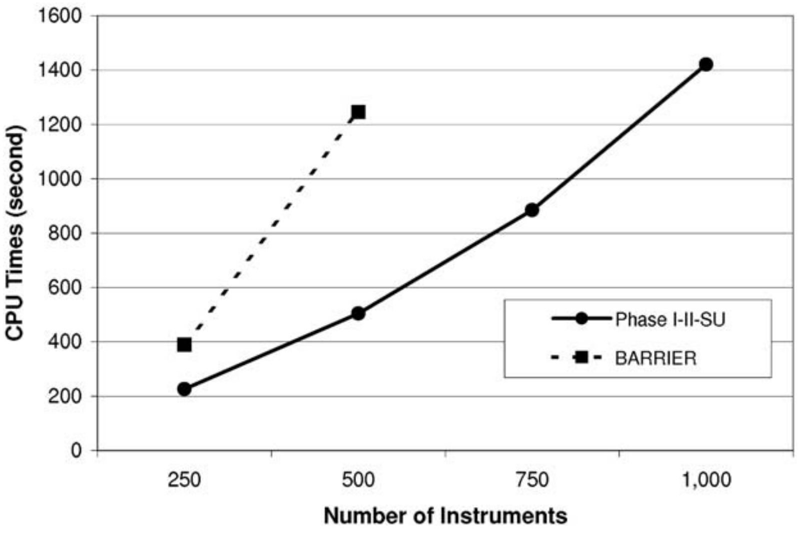

Fig. 3 CPU times for Phase I–II-SU and BARRIER for 50,000-scenario problems

by TPM-SU and BARRIER are depicted in Fig. 4 together with those for the Phase I–II-SU procedure. Similar to the experiment above (see Fig. 3), BARRIER failed to produce solutions for the last three instances due to excessive memory requirements. Among the 17 instances that BARRIER solved, TPM-SU consumed less CPU times than BARRIER did for 10 instances. Considering these 10 instances, the differences between the CPU times for TPM-SU and BARRIER lie in the range [17, 556] with an average of 255.8 seconds. On the other hand, for the other 7 instances where BARRIER performed better, the CPU time differences vary from 7 to 113 seconds with an average of 43.6 seconds. Again, the Phase I–II Method displayed a competitively much smaller effort, ranging within [9, 97] seconds with an average saving of 1749.6 seconds over the Barrier method, while yielding zero optimality gaps for all the 20 instances.

In summary, based on the above results, we recommend the Phase I–II Method for implementation, particularly when the problem size is relatively large or when time is at a premium.

4.2 Experiments using real-world examples

In addition to the above study on randomly generated test instances, we also experimented with several real-world problems. We initially examined two instances compiled by Meucci [20, 21]. The first instance, denoted STOCK, has four international stock indices, the American S&P500, the British FTSE100, the French CAC40, and the German DAX, and the second, denoted BOND, has four US Treasury bonds with 2, 5, 10, and 30 years to maturity, respectively. STOCK consists of 10,000 scenarios generated from prior and posterior distributions (denoted by STOCK-PR and STOCK-PO) using Monte Carlo simulation, while BOND has 50,000 scenarios generated from each of two respective distributions, denoted by BOND-PR and BOND-

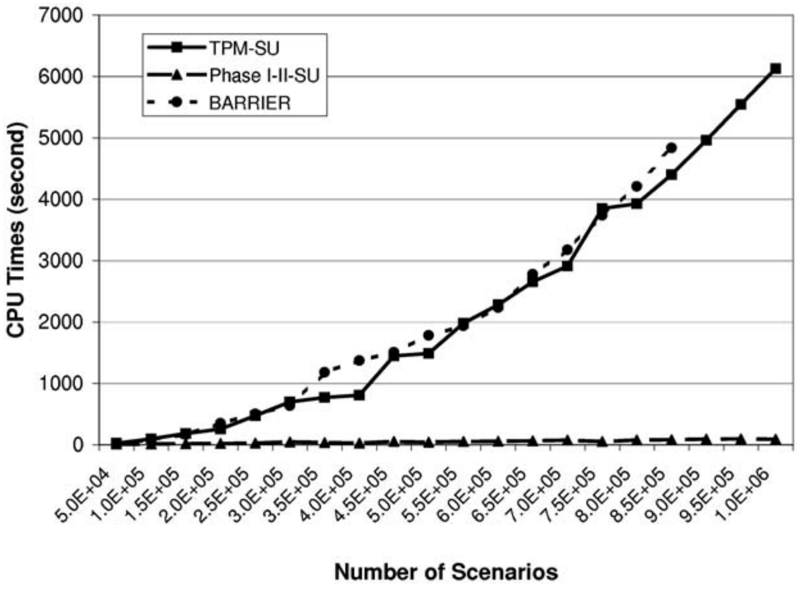

Fig. 4 CPU times for Phase I–II-SU and BARRIER for 10-instrument problems

PO. (See Meucci [20, 21] for details on the related data.) Noting that there are only four instruments, we reduced the stopping tolerance to 10^{-6} for PBA (otherwise, this method performed excessive iterations without sufficient improvements). Again, the confidence level was set at $\alpha = 0.95$.

Table 3 presents the results obtained. Here, CPU and ITR are defined as before, and the optimal objective values are displayed under the corresponding problem's name. Observe that the first phase itself of all the TPM procedures yielded optimal objective values for all the problems, whereas PBA produced the objective values 0.0508531, 0.0508752, 0.00489615, and 0.00467908 for STOCK-PR, STOCK-PO, BOND-PR, and BOND-PO, as compared against the optimal values 0.0481902, 0.0481902, 0.00489605, and 0.00467892, respectively. For STOCK-PR/PO, TPM outperformed PBA-SIM as well as the stand-alone simplex and barrier optimizers in terms of the CPU time required to solve these problems. For example, when solving STOCK-PR, the TMP-SD procedure consumed 22.8, 25.7, and 59.7% of the CPU times required by PBA-SIM, SIMPLEX, and BARRIER, respectively. On the other hand, PBA-SIM displayed a competitive performance when applied to the BOND problems. For example, it consumed the least CPU time for BOND-PR, while no significant difference was observed between TPM and PBA-SIM for BOND-PO. However, as mentioned above, TPM found an optimal solution within the first phase itself. Observe, for example, that the first phase of TPM-SU consumed 0.14, 0.15, 2.23, and 0.58 seconds for solving STOCK-PR, STOCK-PO, BOND-PR, and BOND-PO, respectively. These computational efforts are 7.3–25.3% of those required by the best-performing procedure among PBA-SIM, SIMPLEX, and BARRIER.

As mentioned earlier, this type of problem could be augmented to include the following additional constraint that restricts the expected return to be at least equal to

Table 3 Computational results for real-world examples

Problem (Optimal CVaR)		STOCK-PR (0.0481902)	STOCK-PO (0.0481902)	BOND-PR (0.00489605)	BOND-PO (0.00467892)
CPU	TPM-SD	0.77	0.77	10.37	8.1
	Phase I–II-SD	0.38	0.38	4.22	2.07
	TPM-SU	0.68	0.72	10.28	7.84
	Phase I–II-SU	0.3	0.33	3.71	2.07
	TPM-MBFGS	0.73	0.59	10.29	7.5
	Phase I–II-MBFGS	0.35	0.29	3.98	2.12
	PBA-SIM	3.38	3.24	8.86	7.93
	PBA	1.25	1.22	0.04	0.04
	SIMPLEX	3	3.66	57.42	57.52
	BARRIER	1.29	1.3	12.03	16.37
ITR	TPM-SD	35	34	5	5
	TPM-SU	35	34	5	5
	TPM-MBFGS	35	8	5	7
	PBA-SIM	779	748	127	128
	SIMPLEX	1854	2000	7541	7510

a given target return, r:

$$\sum_{i=1}^{n} m_i x_i \geq r, \tag{14}$$

where m_i is the average of the scenario returns for instrument i, $\forall i = 1, \dots, n$. Meucci [20] used 31 values of r that evenly divide $[0.00096, 0.00105]$, and 51 values of r that evenly divide $[0.0015, 0.005]$. As shown at the end of this section, re-optimizing with a different value of r can be efficiently handled via sensitivity analysis after solving one problem. Therefore, we began by considering the solution of STOCK and BOND using a single value of r. In particular, we used the left interval bounds $r = 0.00096$ and $r = 0.0015$ for STOCK and BOND, respectively.

For implementing the TPM and PBA-SIM procedure, we executed the NDO phases without the constraint (14), then determined an advanced basis as before, and finally added this constraint (14) to the LP formulation in the final phase. Table 4 reports the resulting overall CPU time and the number of simplex iterations. Among the implemented TPM procedures, TPM-SU required the fewest simplex iterations and consumed the least CPU time except for BOND-PR. Furthermore, when compared with the stand-alone simplex algorithm, the TPM-SU procedure consumed 15.0–31.9% of the CPU time required by SIMPLEX. When compared with BARRIER, the Three-Phase Method consumed smaller CPU times for STOCK, whereas BARRIER yielded a better performance for BOND. Although PBA-SIM revealed the best performance in terms of the CPU time when solving BOND-PR, it was not as competitive for the other problems.

Table 4 Computational results for examples with target return constraint

Problem (Optimal CVaR)		STOCK-PR (0.0497455)	STOCK-PO (0.0499127)	BOND-PR (0.0079878)	BOND-PO (0.00695251)
CPU	TPM-SD	0.96	1.25	22.14	23.56
	TPM-SU	0.94	1.21	23.1	10.01
	TPM-MBFGS	0.94	1.36	21.27	23.68
	PBA-SIM	2.63	3.45	20.91	24.92
	SIMPLEX	6.28	5.02	72.36	62.09
	BARRIER	1.34	3.22	13.47	13.95
ITR	TPM-SD	154	259	955	1583
	TPM-SU	154	259	887	150
	TPM-MBFGS	154	270	955	1599
	PBA-SIM	551	789	1364	1861
	SIMPLEX	2172	2124	8125	7537

Recall that Meucci also experimented with additional 30 and 50 target return values for STOCK and BOND, respectively. Since an optimal solution is available from the above experiment, we can sequentially use the corresponding optimal basis as an advanced starting basis for solving the revised problems in which the target values are modified. Performing this sensitivity-update, the solution times required for solving 30 additional STOCK-PR and STOCK-PO problems were 2.1 and 1.84 seconds, respectively, and those for solving 50 additional BOND-PR and BOND-PO problems were 47.1 and 36.3 seconds, respectively.

Finally, as observed above, the Phase I–II Method can be beneficially employed, particularly for problems having a large number of scenarios. To investigate this observation in the context of another real-world problem, we employed the Phase I–II-SU and BARRIER procedures to solve a credit risk problem having three instruments and 500,000 scenarios. (We are not able to disclose further information about the data because it is confidential.) For this instance, the Phase I–II Method attained the optimal value and consumed 44 seconds. However, the barrier method required 3309 seconds, which is still larger than the CPU time consumed by TPM-SU (1666 seconds).

5 Summary and conclusions

In this paper, we have considered a portfolio optimization problem by minimizing the conditional value-at-risk. Noting that this is essentially a nondifferentiable optimization problem (NDO), although it can be formulated as a large-scale linear program, we have proposed a two-phase method and an augmented three-phase method comprised of an NDO procedure (first two phases) coordinated with the simplex algorithm (the third phase). In the first phase, nondifferentiable points are obviated by perturbing the solution at such points to differentiable solutions in the relative vicinity. In this way, we are able to exploit descent-based differentiable optimization techniques. The second phase employs a variable target value NDO procedure, which uses a deflected subgradient search direction along with a step size determined by a suitable target value, and where the next iterate is computed by sequentially projecting the current iterate onto a recent set of Polyak-Kelley cutting planes. This constitutes the Phase I–II Method, which is infinitely convergent to an optimal solution. In addition, the Three-Phase Method achieves finite convergence by employing an advanced crash-basis that is prescribed based on the solution resulting from the first two phases, and then resorting to the simplex algorithm in the third phase.

A computational study was conducted to test the proposed procedures using randomly generated as well as real-world problems. In addition, for benchmarking purposes, we implemented a variant of the proximal bundle algorithm (PBA), and also the simplex and interior point barrier solvers within the commercial package CPLEX 9.0. The proposed Three-Phase Method outperformed the stand-alone simplex algorithm (implemented in CPLEX 9.0) as well as PBA. On average, it consumed 16.5–67.0% of the CPU time required for the simplex algorithm to solve the test problems. Furthermore, as the number of scenarios increases, the Three-Phase Method displayed a preferable performance when compared with the interior point solver. Also, we remark that the Phase I–II Method produced optimal solutions while requiring only 1.7–4.5% and 9.9–61.0% of the CPU effort as compared with the stand-alone simplex and interior point optimizers, respectively. This relative advantage was observed to sharply increase as the number of scenarios and instruments increases, which was evident on both the simulated and the real-world problem instances. Hence, we recommend the implementation of the Phase I–II Method in practice, particularly for large-scale and/or time-sensitive problems.

As observed in the computational result, Phase III may consume a considerable amount of effort, especially when the number of instruments is large. Possible approaches include employing cutting plane or relaxation strategies, or deriving competitive warm-start interior point implementations, and these can be investigated in a future study.

Acknowledgements This research has been supported in part by the *Air Force Office of Scientific Research* under Grant No. F49620-03-1-0377, and by the *National Science Foundation* under Grant Nos. DMI-0552676 and DMI-0457473. The authors also thank the two referees for their constructive comments that have improved the presentation in this paper.

Appendix A: Proof of Lemma 1

Using the convexity of $f^j(\cdot)$, $v > 0$, and $\lambda^j \in [0,1]$, we have that

$$\begin{aligned}
&\widetilde{F}_\alpha(\mathbf{x}^k, \zeta^k) + (\boldsymbol{\xi}^k)^T(\mathbf{x}-\mathbf{x}^k) + g^k(\zeta - \zeta^k) \\
&\quad = \zeta^k + v \sum_{j \in J_+^k} \left[f^j(\mathbf{x}^k) - \zeta^k \right] + v \sum_{j \in J_+^k} \nabla f^j(\mathbf{x}^k)^T(\mathbf{x}-\mathbf{x}^k) \\
&\qquad + v \sum_{j \in J_0^k} \lambda^j \nabla f^j(\mathbf{x}^k)^T(\mathbf{x}-\mathbf{x}^k)
\end{aligned}$$

$$
\begin{aligned}
&+\zeta-\zeta^k-v|J_+^k|(\zeta-\zeta^k)-v(\zeta-\zeta^k)\sum_{j\in J_0^k}\lambda^j\\
&=\zeta+v\sum_{j\in J_+^k}\left[f^j(\mathbf{x}^k)+\nabla f^j(\mathbf{x}^k)^T(\mathbf{x}-\mathbf{x}^k)-\zeta\right]\\
&\quad+v\sum_{j\in J_0^k}\lambda^j\left[f^j(\mathbf{x}^k)+\nabla f^j(\mathbf{x}^k)^T(\mathbf{x}-\mathbf{x}^k)-\zeta\right]\\
&\le\zeta+v\sum_{j\in J_+^k}\left[f^j(\mathbf{x})-\zeta\right]+v\sum_{j\in J_0^k}\lambda^j\left[f^j(\mathbf{x})-\zeta\right]\\
&\le\zeta+v\sum_{j\in J}\max\{f^j(\mathbf{x})-\zeta,0\}\\
&=\widetilde{F}_\alpha(\mathbf{x},\zeta).
\end{aligned}
$$
□

Appendix B: Proof of Theorem 1

Under the assumption of the theorem, defining the index sets as in Lemma 1, we have that

$$\widetilde{F}_\alpha(\mathbf{x},\zeta)=\zeta+v\textstyle\sum_{j\in J_+^k}[f^j(\mathbf{x})-\zeta],\quad\forall(\mathbf{x},\zeta)\in N_\varepsilon(\mathbf{x}^k,\zeta^k).$$

The result now follows by a direct computation of $\nabla\widetilde{F}_\alpha(\mathbf{x}^k,\zeta^k)$, noting that $J_0^k=\emptyset$. □

Appendix C: Three-Phase method

Phase I: PT

Step 0 Select termination parameters $\varepsilon_0=10^{-6}$ and $K_I=100$. Compute $P^*=\{p\in\{1,\dots,n\}:\widetilde{F}_\alpha(\mathbf{e}_p,0)=\text{minimum}_{i=1,\dots,n}\{\widetilde{F}_\alpha(\mathbf{e}_i,0)\}\}$, where $\mathbf{e}_i\in R^n$ is a unit coordinate vector that has one in the ith position and zeros otherwise. Let $\mathbf{x}^0\in R^n$ be a vector such that x_i^0 is equal to $1/|P^*|$ for $i\in P^*$, 0 otherwise. (Alternatively, we could try $\mathbf{x}^0=\mathbf{e}_p$, where $p\in\text{argmin}_{i=1,\dots,n}\{\widetilde{F}_\alpha(\mathbf{e}_i,0)\}$.) Set the initial iterate as $\mathbf{x}_\zeta^1\equiv((\mathbf{x}^0)^T,0)^T$. (If $\widetilde{F}$ is nondifferentiable at this point, perturb $\mathbf{x}_\zeta^1$ by calling Subroutine PT to find a differentiable point.) Set the iteration counter $k=1$ and the restarting indicator *RESET* $=0$.

Step 1 Compute $\widetilde{F}^k\equiv\widetilde{F}_\alpha(\mathbf{x}_\zeta^k)$ and $\boldsymbol{\xi}_g^k\equiv((\boldsymbol{\xi}^k)^T,g^k)^T$ via (4) and (8). If $\|\boldsymbol{\xi}_g^k\|<\varepsilon_0$, terminate the procedure with the current solution. If $k=K_I$, then put the initial target value $w_1=\widetilde{F}^{K_I}-0.6(\widetilde{F}^1-\widetilde{F}^{K_I})$, reset $\mathbf{x}_\zeta^1=\mathbf{x}_\zeta^{K_I}$, and go to Phase II. Otherwise, proceed to Step 2.

Step 2 If *RESET* $=0$ or SD is employed, put $\mathbf{d}^k=-\boldsymbol{\xi}_g^k$. Else, compute the next search direction as follows according to the employed direction strategy:

SU: $$\mathbf{d}^k=-\boldsymbol{\xi}_g^k+\frac{(\boldsymbol{\xi}_g^k)^T(\mathbf{q}^k-\mathbf{d}^{k-1})}{(\mathbf{d}^{k-1})^T\mathbf{q}^k}\mathbf{d}^{k-1}$$

$$\text{MBFGS:} \qquad \mathbf{d}^k = -\frac{(\mathbf{p}^k)^T\mathbf{q}^k}{(\mathbf{q}^k)^T\mathbf{q}^k}\boldsymbol{\xi}_g^k - \left(2\frac{(\mathbf{p}^k)^T\boldsymbol{\xi}_g^k}{(\mathbf{p}^k)^T\mathbf{q}^k} - \frac{(\mathbf{q}^k)^T\boldsymbol{\xi}_g^k}{(\mathbf{q}^k)^T\mathbf{q}^k}\right)\mathbf{p}^k + \frac{(\mathbf{p}^k)^T\boldsymbol{\xi}_g^k}{(\mathbf{q}^k)^T\mathbf{q}^k}\mathbf{q}^k,$$

where $\mathbf{p}^k = \mathbf{x}_\zeta^k - \mathbf{x}_\zeta^{k-1}$ and $\mathbf{q}^k = \boldsymbol{\xi}_g^k - \boldsymbol{\xi}_g^{k-1}$.

Step 3 If SU or MBFGS is employed and $(\mathbf{d}^k)^T\boldsymbol{\xi}_g^k \geq 0$, put $RESET = 0$ and return to Step 2. Else, perform a single quadratic-fit line-search along the direction $\mathbf{d}^k$ (and project onto $\mathbf{X}$ using the PROJECTION scheme of Sect. 2, if necessary) to obtain the next iterate $\mathbf{x}_\zeta^{k+1}$. If $\widetilde{F}$ is nondifferentiable at $\mathbf{x}_\zeta^{k+1}$, perturb it to find a differentiable point as described in Sect. 3. Increment $k \leftarrow k+1$ and $RESET \leftarrow RESET + 1$. If $RESET = n+1$, put $RESET = 0$. Return to Step 1.

Phase II: M-VTVM

Step 0 (Initialization) Select algorithmic parameters $\varepsilon_0 = 10^{-6}$, $\varepsilon = 0.1$, $K_{II} = 1000$, $\rho = 0.8$, $\sigma = 0.15$, $\eta = 0.75$, $r = 0.1$, $\overline{r} = 1.1$, $\overline{\tau} = 75$, and $\overline{\gamma} = 20$. Set an initial iterate for M-VTVM as $\mathbf{x}_\zeta^1 = \mathbf{x}_\zeta^{K_I}$, and compute $\widetilde{F}^1$ and $\boldsymbol{\xi}_g^1$. Initialize incumbent values $\overline{F} = \widetilde{F}^1$, $\overline{\mathbf{x}}_\zeta = (\mathbf{x}^1, \zeta^1)$, and $\overline{\boldsymbol{\xi}}_g = \boldsymbol{\xi}_g^1$. Put $\varepsilon_1 = \sigma(\widetilde{F}^1 - w_1)$. Set indicators and counters $RESET = 1$, $\Delta = 0$, $\tau = 0$, $\gamma = 0$, $k_0 = 1$, $k = 1$, and $l = 1$.

Step 1 (GPKC Strategy) Call Subroutine GPKC to compute $\mathbf{x}_\zeta^{k+1}$ (replace $\mathbf{x}_\zeta^{k+1} \leftarrow P_{\mathbf{X}}(\mathbf{x}_\zeta^{k+1})$ if $\mathbf{x}_\zeta^{k+1} \notin \mathbf{X}$). Increment $\tau \leftarrow \tau + 1$ and $k \leftarrow k+1$. Compute $\widetilde{F}^k$ and $\boldsymbol{\xi}_g^k$. If $\|\boldsymbol{\xi}_g^k\| < \varepsilon_0$, terminate the algorithm. If $k > K_{II}$, go to Phase III with $\overline{\mathbf{x}}_\zeta$. Put $RESET = 0$.

Step 2 If $\widetilde{F}^k < \overline{F}$, update incumbent values as $\overline{F} = \widetilde{F}^k$, $\overline{\mathbf{x}}_\zeta = (\mathbf{x}^k, \zeta^k)$, $\overline{\boldsymbol{\xi}}_g = \boldsymbol{\xi}_g^k$, and set $\Delta \leftarrow \Delta + \overline{F} - \widetilde{F}^k$ and $\gamma = 0$. Otherwise, increment $\gamma \leftarrow \gamma + 1$. If $\overline{F} \leq w_l + \varepsilon_l$, proceed to Step 3. If $\gamma \geq \overline{\gamma}$ or $\tau \geq \overline{\tau}$, go to Step 4. Else, return to Step 1.

Step 3 Compute $w_{l+1} = \overline{F} - \max\{\varepsilon_l + \eta\Delta,\ r|\overline{F}|\}$, and update $\varepsilon_{l+1} = \max\{(\overline{F} - w_{l+1})\sigma, \varepsilon\}$. If $k \leq 500$, update $\eta \leftarrow 2\eta$; otherwise, set $\eta = 0.75$. If $r|\overline{F}| > \varepsilon_l + \eta\Delta$, update $r \leftarrow r/\overline{r}$. Put $\tau = 0$, $\Delta = 0$, and increment $l \leftarrow l+1$. Return to Step 1.

Step 4 Compute $w_{l+1} = (\overline{F} - \varepsilon_l + w_l)/2$, and update $\varepsilon_{l+1} = \max\{(\overline{F} - w_{l+1})\sigma, \varepsilon\}$. If $k \leq 500$, update $\eta \leftarrow \max\{\eta/2, 0.75\}$; otherwise, set $\eta = 0.75$. If $\gamma \geq \overline{\gamma}$, set $\overline{\gamma} = \min\{\overline{\gamma} + 10, 50\}$. If $w_{l+1} - w_l \leq 0.1$, then set $\overline{\beta} \leftarrow \max\{\overline{\beta}/2, \varepsilon_0\}$. Put $\gamma = 0$, $\tau = 0$, $\Delta = 0$, and $l \leftarrow l+1$. Reset $\mathbf{x}^k = \overline{\mathbf{x}}$, $\widetilde{F}^k = \overline{F}$, and $\boldsymbol{\xi}_g^k = \overline{\boldsymbol{\xi}}_g$. Put $RESET = 1$ and $k_0 = k - 1$. Return to Step 1.

Subroutine GPKC:

Step i: Let $\delta = \min\{k - k_0, 4\}$, and put $\widehat{F} = \widetilde{F}^k - \rho(\widetilde{F}^k - w_l)$. Set $\Psi_1 = (\widetilde{F}^k - \widehat{F})/\|\boldsymbol{\xi}_g^k\|^2$, $\Psi_2 = 0$, and $\mathbf{x}_\zeta^{k+1} = \mathbf{x}_\zeta^k - \Psi_1\boldsymbol{\xi}_g^k$. If $RESET = 1$, put $k_0 = k$ and exit the subroutine.

Step ii: Compute $\theta_1 = \widehat{F} - \widetilde{F}^k + (\mathbf{x}_\zeta^k)^T\boldsymbol{\xi}_g^k$ and $\theta_2 = \widehat{F} - \widetilde{F}^{k-1} + (\mathbf{x}_\zeta^{k-1})^T\boldsymbol{\xi}_g^{k-1}$. If $(\mathbf{x}_\zeta^{k+1})^T\boldsymbol{\xi}_g^{k-1} \leq \theta_2$, go to Step v.

Step iii: Put $\Psi_2 = [(\mathbf{x}_\zeta^k)^T \boldsymbol{\xi}_g^{k-1} - \theta_2]/\|\boldsymbol{\xi}_g^{k-1}\|^2$ and $\mathbf{x}_\zeta^{k+1} = \mathbf{x}_\zeta^k - \Psi_2\boldsymbol{\xi}_g^{k-1}$. If $(\mathbf{x}_\zeta^{k+1})^T\boldsymbol{\xi}_g^k \le \theta_1$, go to Step v.

Step iv: Compute $\Psi_0 = \|\boldsymbol{\xi}_g^{k-1}\|^2\|\boldsymbol{\xi}_g^k\|^2 - [(\boldsymbol{\xi}_g^{k-1})^T\boldsymbol{\xi}_g^k]^2$. If $\Psi_0 < \varepsilon_0$, put $\mathbf{x}_\zeta^{k+1} = \mathbf{x}_\zeta^k - \Psi_1\boldsymbol{\xi}_g^k$ and exit the subroutine. Otherwise, compute $\Psi_1 = [\|\boldsymbol{\xi}_g^{k-1}\|^2(\widehat{F} - \widetilde{F}^k) - ((\boldsymbol{\xi}_g^{k-1})^T\boldsymbol{\xi}_g^k)(\theta_2 - (\mathbf{x}_\zeta^k)^T\boldsymbol{\xi}_g^{k-1})]/\Psi_0$ and $\Psi_2 = [\|\boldsymbol{\xi}_g^k\|^2(\theta_2 - (\mathbf{x}_\zeta^k)^T\boldsymbol{\xi}_g^{k-1}) - ((\boldsymbol{\xi}_g^{k-1})^T\boldsymbol{\xi}_g^k)(\widehat{F} - \widetilde{F}^k)]/\Psi_0$. Put $\mathbf{x}_\zeta^{k+1} = \mathbf{x}_\zeta^k + \Psi_1\boldsymbol{\xi}_g^k + \Psi_2\boldsymbol{\xi}_g^{k-1}$, and proceed to Step v.

Step v: If $\delta = 1$, exit the subroutine. Otherwise, put $i = 2$ and proceed to Step vi.

Step vi: Put $\theta_3 = \widehat{F} - \widetilde{F}^{k-i} + (\mathbf{x}_\zeta^{k-i})^T\boldsymbol{\xi}_g^{k-i}$ and $\Psi_3 = [(\mathbf{x}_\zeta^{k+1})^T\boldsymbol{\xi}_g^{k-i} - \theta_3]/\|\boldsymbol{\xi}_g^{k-i}\|^2$. If $\Psi_3 \le 0$, go to Step viii. Else, proceed to Step vii.

Step vii: Put $\widehat{\mathbf{x}}_\zeta^k = \mathbf{x}_\zeta^{k+1} - \Psi_3\boldsymbol{\xi}_g^{k-i}$. If $(\widehat{\mathbf{x}}_\zeta^k)^T\boldsymbol{\xi}_g^k \le \theta_1$ and $(\widehat{\mathbf{x}}_\zeta^k)^T\boldsymbol{\xi}_g^{k-1} \le \theta_2$, put $\mathbf{x}_\zeta^{k+1} = \widehat{\mathbf{x}}_\zeta^k$.

Step viii: If $i = \delta$, exit the subroutine. Else, increment $i \leftarrow i + 1$, and return to Step vi.

Phase III: SIMPLEX

Step 1 Given $\overline{\mathbf{x}}_\zeta$, find a feasible solution $\theta \equiv (\overline{\mathbf{z}}, \overline{\mathbf{s}}, \overline{\mathbf{x}}, \overline{\zeta})$ as in (13).

Step 2 Crash variables into the basis in the order of $\{\zeta; z_j : \overline{z}_j > 0; s_j : \overline{s}_j > 0; x_j$ ordered in nonincreasing order of $\overline{x}_j, j = 1, \dots, n\}$. Run the simplex optimizer of CPLEX from this advanced basis specification.

Appendix D: Proximal bundle algorithm (PBA)

In the proximal bundle method, the objective function is approximated by a quadratic convex function whose curvature is determined by the coefficient of the quadratic term, the so-called *proximity weight*. This method works with the current subgradient and an aggregate representation of previously generated subgradients, which renders it suitable for solving large-scale NDO problems by trading-off storage and effort per iteration versus the speed of convergence (see Kiwiel [13, 14]). Kiwiel [14] proposes a variable proximity weight method that updates the weight as necessary. However, in some preliminary experiments using our test problems, this variable proximity weight method did not perform well. Specifically, it failed to increase the proximity weight whenever a greater weight (i.e., a smaller step size) was required for attaining descent. Based on our preliminary search for proximity weights, we chose a fixed proximity weight of 100, which yields a relatively good performance for our test problems. Accordingly, we set the stopping tolerance as 10^{-10}, which is smaller than that implemented by Kiwiel [14]. Other algorithmic parameter values were selected as in the numerical study of [14]. This improved version of PBA is detailed below.

Step 0 Select a termination parameter $\varepsilon = 10^{-10}$, an improvement parameter $m_L = 0.1$, and a proximity weight $u = 100$. Set the initial iterate $\mathbf{x}_\zeta^1$ as in the PT phase of the proposed procedure in Sect. 3. Compute $\widehat{\boldsymbol{\xi}}_g^1 \in \partial\widetilde{F}_\alpha(\widehat{\mathbf{x}}_\zeta^1)$. Put $\widehat{\mathbf{x}}_\zeta^1 = \mathbf{x}_\zeta^1$, $\overline{\boldsymbol{\xi}}_g^0 = \widehat{\boldsymbol{\xi}}_g^1$, $G^1 = \widetilde{F}^1$, $\overline{G}^1 = \widetilde{F}^1$, $\omega^1 = 0$, and $\overline{\omega}^0 = 0$. Set the iteration counter $k = 1$.

Step 1 Solve the quadratic subproblem

$$
\begin{aligned}
\textbf{QS:} \quad & \text{Minimize} && y + u\|\mathbf{d}\|^2/2 && \text{(15a)}\\
& \text{subject to} && -\omega^k + (\widehat{\boldsymbol{\xi}}_g^k)^T \mathbf{d} \le y, && \text{(15b)}\\
& && -\overline{\omega}^k + (\overline{\boldsymbol{\xi}}_g^{k-1})^T \mathbf{d} \le y, && \text{(15c)}\\
& && \mathbf{d} \in \mathbf{D}, && \text{(15d)}
\end{aligned}
$$

where $\mathbf{D} = \{(\mathbf{d_x}, d_\zeta) \in R^{n+1} : \mathbf{e}^T \mathbf{d_x} = 0,\ \mathbf{d_x} \ge -\mathbf{x}^k\}$. Let $(\mathbf{d}^k, y^k)$ be a solution to QS with Lagrange multipliers $(\lambda^k, \overline{\lambda}^k)$ associated with (15b)–(15c). Put $\overline{\boldsymbol{\xi}}_g^k = \lambda^k \widehat{\boldsymbol{\xi}}_g^k + \overline{\lambda}^k \overline{\boldsymbol{\xi}}_g^{k-1}$, $\overline{G}_a^k = \lambda^k G^k + \overline{\lambda}^k \overline{G}^k$, and $\overline{\omega}_a^k = \lambda^k \omega^k + \overline{\lambda}^k \overline{\omega}^k$. Let $\overline{G}_a^k(\mathbf{x}_\zeta) = \overline{G}_a^k + (\overline{\boldsymbol{\xi}}_g^k)^T(\mathbf{x}_\zeta - \mathbf{x}_\zeta^k)$.

Step 2 If $y^k > -\varepsilon$, terminate. Otherwise, proceed to Step 3.

Step 3 Set $\widehat{\mathbf{x}}_\zeta^{k+1} = \mathbf{x}_\zeta^k + \mathbf{d}^k$. If $\widetilde{F}_\alpha(\widehat{\mathbf{x}}_\zeta^{k+1}) \le \widetilde{F}^k + m_L y^k$, then put $\mathbf{x}_\zeta^{k+1} = \widehat{\mathbf{x}}_\zeta^{k+1}$. Otherwise, set $\mathbf{x}_\zeta^{k+1} = \mathbf{x}_\zeta^k$.

Step 4 Put $\overline{G}^{k+1} = \overline{G}_a^k(\mathbf{x}_\zeta^{k+1})$. Compute $\widehat{\boldsymbol{\xi}}_g^{k+1} \in \partial \widetilde{F}_\alpha(\widehat{\mathbf{x}}_\zeta^{k+1})$ and $G^{k+1} = \widetilde{F}_\alpha(\widehat{\mathbf{x}}_\zeta^{k+1}) + (\widehat{\boldsymbol{\xi}}_g^{k+1})^T(\mathbf{x}_\zeta^{k+1} - \widehat{\mathbf{x}}_\zeta^{k+1})$. Set $\omega^{k+1} = \widetilde{F}^{k+1} - G^{k+1}$ and $\overline{\omega}^{k+1} = \widetilde{F}^{k+1} - \overline{G}^{k+1}$.

Step 5 Increment $k \leftarrow k + 1$ and return to Step 1.

References

1. Andersson, F., Mausser, H., Rosen, D., Uryasev, S.: Credit risk optimization with conditional value-at-risk criterion. Math. Program. **89**(2), 273–291 (2001)
2. Artzner, P., Delbaen, F., Eber, J.M., Heath, D.: Coherent measures of risk. Math. Finance **9**(3), 203–228 (1999)
3. Barahona, F., Anbil, R.: The volume algorithm: Producing primal solutions with a subgradient method. Math. Program. **87**(3), 385–399 (2000)
4. Barr, D.R., Slezak, N.L.: A comparison of multivariate normal generators. Commun. ACM **15**(12), 1048–1049 (1972)
5. Bazaraa, M.S., Sherali, H.D., Shetty, C.M.: Nonlinear Programming: Theory and Algorithms, 3rd edn. Wiley, New York (2006)
6. Bitran, G., Hax, A.: On the solution of convex knapsack problems with bounded variables. In: Proceedings of IXth International Symposium on Mathematical Programming, Budapest, pp. 357–367 (1976)
7. Brännlund, U.: On relaxation methods for nonsmooth convex optimization. Ph.D. Dissertation, Department of Mathematics, Royal Institute of Technology, Stockholm, Sweden (1993)
8. Butenko, S., Golodnikov, A., Uryasev, S.: Optimal security liquidation algorithms. Comput. Optim. Appl. **32**(1), 9–27 (2005)
9. Goffin, J.L., Kiwiel, K.C.: Convergence of a simple subgradient level method. Math. Program. **85**(1), 207–211 (1999)
10. Held, M., Wolfe, P., Crowder, H.: Validation of subgradient optimization. Math. Program. **6**(1), 62–88 (1974)
11. Jabr, R.A.: Robust self-scheduling under price uncertainty using conditional value-at-risk. IEEE Trans. Power Syst. **20**(4), 1852–1858 (2005)
12. Jabr, J., Baíllo, Á., Ventosa, M., García-Alcalde, A., Perán, F., Relaño, G.: A medium-term integrated risk management model for a hydrothermal generation company. IEEE Trans. Power Syst. **20**(3), 1379–1388 (2005)
13. Kiwiel, K.C.: Methods of Descent for Nondifferentiable Optimization. Lecture Notes in Mathematics, vol. 1133. Springer, New York (1985)

14. Kiwiel, K.C.: Proximity control in bundle methods for convex nondifferentiable minimization. Math. Program. **46**(1), 105–122 (1990)
15. Krokhmal, P., Uryasev, S.: A sample-path approach to optimal position liquidation. Ann. Oper. Res. **152**(1), 193–225 (2007)
16. Krokhmal, P., Uryasev, S., Zrazhevsky, G.: Risk management for hedge fund portfolios: A comparative analysis of linear portfolio rebalancing strategies. J. Altern. Invest. **5**(1), 10–29 (2002)
17. Krokhmal, P., Palmquist, J., Uryasev, S.: Portfolio optimization with conditional value-at-risk objective and constraints. J. Risk **4**(2), 11–27 (2002)
18. Lim, C., Sherali, H.D.: A trust region target value method for optimizing nondifferentiable Lagrangian duals of linear programs. Math. Methods Oper. Res. **64**(1), 33–53 (2006)
19. Lim, C., Sherali, H.D.: Convergence and computational analyses for some variable target value and subgradient deflection methods. Comput. Optim. Appl. **34**(3), 409–428 (2006)
20. Meucci, A.: Beyond Black-Litterman: Views on non-normal markets. Soc. Sci. Res. Netw. http://papers.ssrn.com (2005)
21. Meucci, A.: Beyond Black-Litterman in practice: a five-step recipe to input views on non-normal markets. Soc. Sci. Res. Netw. http://papers.ssrn.com (2005)
22. Polyak, B.T.: A general method of solving extremum problems. Sov. Math. **8**(3), 593–597 (1967)
23. Polyak, B.T.: Minimization of unsmooth functionals. U.S.S.R. Comput. Math. Math. Phys. **9**(3), 14–39 (1969)
24. Rockafellar, R.T., Uryasev, S.: Optimization of conditional value-at-risk. J. Risk **2**(3), 21–41 (2000)
25. Rockafellar, R.T., Uryasev, S.: Conditional value-at-risk for general loss distributions. J. Bank. Finance **26**(7), 1443–1471 (2002)
26. Scheuer, E.M., Stoller, D.S.: On the generation of normal random vectors. Technometrics **4**(2), 278–281 (1962)
27. Shanno, D.F.: Conjugate gradient methods with inexact searches. Math. Oper. Res. **3**(3), 244–256 (1978)
28. Sherali, H.D., Lim, C.: On embedding the volume algorithm in a variable target value method. Oper. Res. Lett. **32**(5), 455–462 (2004)
29. Sherali, H.D., Lim, C.: Enhancing Lagrangian dual optimization for linear programs by obviating nondifferentiability. INFORMS J. Comput. **19**(1), 3–13 (2007)
30. Sherali, H.D., Shetty, C.M.: On the generation of deep disjunctive cutting planes. Nav. Res. Logist. **27**(3), 453–475 (1980)
31. Sherali, H.D., Ulular, O.: Conjugate gradient methods using quasi-Newton updates with inexact line searches. J. Math. Anal. Appl. **150**(2), 359–377 (1990)
32. Sherali, H.D., Choi, G., Tuncbilek, C.H.: A variable target value method for nondifferentiable optimization. Oper. Res. Lett. **26**(1), 1–8 (2000)
33. Sherali, H.D., Choi, G., Ansari, Z.: Limited memory space dilation and reduction algorithms. Comput. Optim. Appl. **19**(1), 55–77 (2001)
34. Uryasef, S.: New variable-metric algorithms for nondifferentiable optimization problems. J. Optim. Theory Appl. **71**(2), 359–388 (1991)
35. Uryasef, S.: Conditional value-at-risk: optimization algorithms and applications. Financ. Eng. News **14**, 1–5 (2000)