Working notes—Not for circulation.

Dominique Orban

A Python implementation of SUPERB

March 10, 2008

Abstract. We describe and discuss an implementation of the mixed interior/exterior-point method for nonlinear programming described in [GOT03a]. The implementation is realized in the NLPy Python framework, `nlpy.sf.net`, for linear and nonlinear optimization.

Contents

1. Problem structure 1
2. Interior-point framework 4
3. The inner iteration 8
4. A Python class for SUPERB 10

1. Problem structure

1.1. Problem statement

We consider the general nonlinear programming problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & c_{\mathcal{E}}(x) = \gamma^{\text{E}} \\ & \gamma^{\text{L}} \leq c_{\mathcal{I}}(x) \leq \gamma^{\text{U}} \\ & x^{\text{L}} \leq x \leq x^{\text{U}}, \end{array} \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $c_{\mathcal{E}} : \mathbb{R}^n \to \mathbb{R}^{n_{\mathcal{E}}}$ and $c_{\mathcal{I}} : \mathbb{R}^n \to \mathbb{R}^{n_{\mathcal{I}}}$ are twice-continuously differentiable functions and where $\gamma^{\text{E}} \in \mathbb{R}^{n_{\mathcal{E}}}$, $\gamma^{\text{L}}, \gamma^{\text{U}} \in \mathbb{R}^{n_{\mathcal{I}}}$ and $x^{\text{L}}, x^{\text{U}} \in \mathbb{R}^n$ are fixed vectors. Fixed variables, if any, are directly eliminated by the code and their values substituted into the expressions where they appear. We define the index sets

$$\mathcal{C}^{\text{L}} = \{i \in \mathcal{I} \mid -\infty < \gamma_i^{\text{L}} \quad \text{and} \quad \gamma_i^{\text{U}} = +\infty\} \tag{1.2a}$$
$$\mathcal{C}^{\text{U}} = \{i \in \mathcal{I} \mid -\infty = \gamma_i^{\text{L}} \quad \text{and} \quad \gamma_i^{\text{U}} < +\infty\} \tag{1.2b}$$
$$\mathcal{C}^{\text{R}} = \{i \in \mathcal{I} \mid -\infty < \gamma_i^{\text{L}} \quad \text{and} \quad \gamma_i^{\text{U}} < +\infty\} \tag{1.2c}$$

and refer to them as the set of *lower constraints*, the set of *upper constraints* and the set of *range constraints* respectively. Note that we have not distinguished linear from nonlinear constraints—possibly,

Dominique Orban: GERAD and École Polytechnique, Montréal, Canada. e-mail: : Dominique.Orban@gerad.ca

one or more constraints $c_i(x)$ with $i \in \mathcal{E} \cup \mathcal{I}$ are linear. For the time being, the only reason to distinguish general constraints from simple bounds is that the two are usually stored in separate data structures when decoding problems written in modern modeling languages, such as SIF [GOT03b] or AMPL [FGK02]. In later refinements, we may wish to treat simple bounds differently, e.g., ensure they are satisfied throughout.

Similarly, we define the sets

$$\mathcal{B}^{\mathrm{L}} = \{1 \le i \le n \mid -\infty < x_i^{\mathrm{L}} < x_i^{\mathrm{U}} = +\infty\} \tag{1.3a}$$
$$\mathcal{B}^{\mathrm{U}} = \{1 \le i \le n \mid -\infty = x_i^{\mathrm{L}} < x_i^{\mathrm{U}} < +\infty\} \tag{1.3b}$$
$$\mathcal{B}^{\mathrm{R}} = \{1 \le i \le n \mid -\infty < x_i^{\mathrm{L}} < x_i^{\mathrm{U}} < +\infty\} \tag{1.3c}$$

and refer to them as the set of *lower bounds*, the set of *upper bounds* and the set of *two-sided bounds* respectively.

Finally, let $\mathcal{C} = \mathcal{C}^{\mathrm{L}} \cup \mathcal{C}^{\mathrm{U}} \cup \mathcal{C}^{\mathrm{R}}$ and $\mathcal{B} = \mathcal{B}^{\mathrm{L}} \cup \mathcal{B}^{\mathrm{U}} \cup \mathcal{B}^{\mathrm{R}}$.

After the decoding stage, a practical implementation will usually "see" the above sets and the variables which they determine. Hence, the indexing in these notes will correspond to simple loops over the appropriate index sets in the implementation. For instance, in Python, the expression $\sum_{i \in \mathcal{E}} c_i(x)$ will correspond to the simple loop

```
ce = 0.0; for i in E: ce += c[i]
```

1.2. Penalty function and elastic problem

From the formulation (1.1), we may apply the transformations given in [GOT03a] to define elastic variables s and t associated to the general constraints and bounds respectively, and the smoothened ℓ_1-penalty function

$$\phi^{\mathrm{P}}(x, s, t; \nu) = f(x) + \sum_{i \in \mathcal{E}} \nu_i \left[c_i(x) - \gamma_i^{\mathrm{E}} + 2 s_i \right] + \sum_{i \in \mathcal{C}} \nu_i s_i + \sum_{i \in \mathcal{B}} \nu_i t_i, \tag{1.4}$$

and obtain the *elastic* problem

$$\begin{aligned}
\underset{x,s,t}{\text{minimize}} \quad & \phi^{\mathrm{P}}(x, s, t; \nu) \\
\text{subject to} \quad & c_i(x) - \gamma_i^{\mathrm{E}} + s_i \ge 0 \quad \text{and} \quad s_i \ge 0 \qquad i \in \mathcal{E} \\
& c_i(x) - \gamma_i^{\mathrm{L}} + s_i \ge 0 \quad \text{and} \quad s_i \ge 0 \qquad i \in \mathcal{C}^{\mathrm{L}} \\
& \gamma_i^{\mathrm{U}} - c_i(x) + s_i \ge 0 \quad \text{and} \quad s_i \ge 0 \qquad i \in \mathcal{C}^{\mathrm{U}} \\
& c_i(x) - \gamma_i^{\mathrm{L}} + s_i \ge 0, \quad \gamma_i^{\mathrm{U}} - c_i(x) + s_i \ge 0 \quad \text{and} \quad s_i \ge 0 \qquad i \in \mathcal{C}^{\mathrm{R}} \\
& \\
& x_i - x_i^{\mathrm{L}} + t_i \ge 0 \quad \text{and} \quad t_i \ge 0 \qquad i \in \mathcal{B}^{\mathrm{L}} \\
& x_i^{\mathrm{U}} - x_i + t_i \ge 0 \quad \text{and} \quad t_i \ge 0 \qquad i \in \mathcal{B}^{\mathrm{U}} \\
& x_i - x_i^{\mathrm{L}} + t_i \ge 0, \quad x_i^{\mathrm{U}} - x_i + t_i \ge 0 \quad \text{and} \quad t_i \ge 0 \qquad i \in \mathcal{B}^{\mathrm{R}}.
\end{aligned} \tag{1.5}$$

Note that a single elastic s_i corresponds to both sides of a range constraint $i \in \mathcal{C}^{\mathrm{R}}$ and a single t_i corresponds to an $i \in \mathcal{B}^{\mathrm{R}}$. The above elastic problem details the structure of each constraint in relation to the index set to which it corresponds, and the penalization associated to them. This problem has $2(n_{\mathcal{E}} + n_{\mathcal{I}}) + |\mathcal{C}^{\mathrm{R}}| + 2(|\mathcal{B}^{\mathrm{L}}| + |\mathcal{B}^{\mathrm{U}}| + |\mathcal{B}^{\mathrm{R}}|) + |\mathcal{B}^{\mathrm{R}}|$ general constraints, linear and nonlinear. Note that for generality, we have considered as many penalty parameters ν_i are there are constraints in (1.5). For simplicity, an implementation may just set $\nu_i = \nu > 0$ for all i.

The Python method to evaluate $\phi^{\mathrm{P}}(x, s, t; \nu)$ is named `evalP` and relies on the method `evalC` which first transforms the constraints. The latter is given in Listing 1.

Listing 1. Constraint Transformation Method

```
    # ======================================================================

    # Reformulate constraints.
    # The "upper" side of a range constraint is pushed at the end of c;
    # If i is in Range, the index of the "upper" side is given by
    #    nConst + Range.index( i )
    def evalC( self, x ):
        c = self.cons( x )
        cR = {}
        for i in self.Eq + self.lowerC:
            c[i] -= self.Lcon[i]
        for i in self.upperC:
            c[i] = self.Ucon[i] - c[i]
        for i in self.Range:
```

We may now define `evalP` as in Listing 2.

Listing 2. Merit Function Evaluation Method

```
    # ======================================================================

    def evalP( self, x, s, t ):
        p = self.obj( x )
        (c, cR) = self.evalC( x )
        # Add contribution from equality constraints
        for i in self.Eq:
            p += self.nuEq * ( c[i] + 2 * s[i] )
        # Add contribution from inequality and range constraints
        for i in self.Ineq + self.Range:
            p += self.nuS * s[i]
        # Add contribution from bound constraints
```

1.3. Derivatives of the penalty function

The first derivatives of (1.4) are given by

$$\nabla_{xst}\phi^{\mathrm{P}}(x, s, t; \nu) = \begin{bmatrix} \nabla_x \phi^{\mathrm{P}}(x, s, t; \nu) \\ \nabla_s \phi^{\mathrm{P}}(x, s, t; \nu) \\ \nabla_t \phi^{\mathrm{P}}(x, s, t; \nu) \end{bmatrix}$$

where

$$\nabla_x \phi^{\mathrm{P}}(x, s, t; \nu) = \nabla f(x) + \sum_{i \in \mathcal{E}} \nu_i \nabla c_i(x), \tag{1.6}$$

$$\nabla_s \phi^{\mathrm{P}}(x, s, t; \nu) = 2 \sum_{i \in \mathcal{E}} \nu_i e_i + \sum_{i \in \mathcal{C}} \nu_i e_i \tag{1.7}$$

and

$$\nabla_t \phi^{\mathrm{P}}(x, s, t; \nu) = \sum_{i \in \mathcal{B}} \nu_i e_i, \tag{1.8}$$

and where each vector e_i is the i-th column of the identity matrix.

It is easy to see that the Hessian matrix of $\phi^{\mathrm{P}}(x, s, t; \nu)$ is given by

$$\nabla^2 \phi^{\mathrm{P}}(x, s, t; \nu) = \begin{bmatrix} \nabla^2 f(x) + \sum_{i \in \mathcal{E}} \nu_i \nabla^2 c_i(x) & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{1.9}$$

In §2, we treat the constraints of the elastic problem (1.5) by means of a logarithmic barrier and explicit its derivatives. The implementation computes the derivatives of the resulting barrier function directly instead of computing the derivatives of ϕ^{P}.

2. Interior-point framework

2.1. A log-barrier merit function

Treating the constraints of (1.5) using a logarithmic barrier are gathering up sums with similar terms, we obtain the *unconstrained* problem

$$\underset{x,s,t}{\text{minimize}} \;\; \phi^{\mathrm{B}}(x,s,t;\mu,\nu), \tag{2.10}$$

where

$$\begin{aligned}\phi^{\mathrm{B}}(x,s,t;\mu,\nu) = \phi^{\mathrm{P}}(x,s,t;\nu) &- \sum_{i\in\mathcal{E}} \mu_i \log(c_i(x) - \gamma_i^{\mathrm{E}} + s_i) - \sum_{i\in\mathcal{E}} \mu_i \log(s_i) \\ &- \sum_{i\in\mathcal{C}^{\mathrm{L}}\cup\mathcal{C}^{\mathrm{R}}} \mu_i \log(c_i(x) - \gamma_i^{\mathrm{L}} + s_i) - \sum_{i\in\mathcal{C}} \mu_i \log(s_i) \\ &- \sum_{i\in\mathcal{C}^{\mathrm{U}}\cup\mathcal{C}^{\mathrm{R}}} \mu_i \log(\gamma_i^{\mathrm{U}} - c_i(x) + s_i) \\ &- \sum_{i\in\mathcal{B}^{\mathrm{L}}\cup\mathcal{B}^{\mathrm{R}}} \mu_i \log(x_i - x_i^{\mathrm{L}} + t_i) - \sum_{i\in\mathcal{B}} \mu_i \log(t_i) \\ &- \sum_{i\in\mathcal{B}^{\mathrm{U}}\cup\mathcal{B}^{\mathrm{R}}} \mu_i \log(x_i^{\mathrm{U}} - x_i + t_i).\end{aligned} \tag{2.11}$$

Again, we choose to use as many barrier parameters μ_i as there are constraints to take the scaling of these constraints into account, keeping in mind that an implementation might very well just pick $\mu_i = \mu$ for all i.

The method to evaluate $\phi^{\mathrm{B}}(x,s,t;\mu,\nu)$ is given in Listing 3.

Listing 3. Barrier Function Evaluation Method

```
1
2       # phi(x, mu) = p(x,s,nu) - mu * Sum log( ci(x) + si )
3       #                        - mu * Sum log( si )
4       #                        - mu * Sum log( bi )
5       def Phi( self, x, s, t ):
6
7           phi = self.evalP( x, s, t )
8           (c, cR) = self.evalC( x )
9           bar = 0.0
10
11          for i in (self.Ineq + self.Eq):
12              bar += math.log( c[i] + s[i] ) + math.log( s[i] )
13          for i in self.Range:
14              bar += math.log( c[i] + s[i] ) + \
15                     math.log( cR[i] + s[i] ) + \
16                     math.log( s[i] )
17
18          for i in self.lowerB:
19              bar += math.log( x[i] - self.Lvar[i] + t[i] ) + math.log( t[i] )
20          for i in  self.upperB:
21              bar += math.log( self.Uvar[i] - x[i] + t[i] ) + math.log( t[i] )
22          for i in self.rangeB:
23              bar += math.log( x[i] - self.Lvar[i] + t[i] ) + \
```

The minimization (2.10) requires to start with a strictly feasible initial guess (x^0, s^0, t^0), which is easily achieved by picking any x^0 and choosing

$$s_i^0 = \begin{cases} \max[0, \gamma_i^{\mathrm{E}} - c_i(x^0)] + \epsilon^{\mathrm{E}}, & i \in \mathcal{E} \\ \max[0, c_i(x^0) - \gamma_i^{\mathrm{L}}] + \epsilon^{\mathrm{CL}}, & i \in \mathcal{C}^{\mathrm{L}} \\ \max[0, \gamma_i^{\mathrm{U}} - c_i(x^0)] + \epsilon^{\mathrm{CU}}, & i \in \mathcal{C}^{\mathrm{U}} \\ \max[0, c_i(x^0) - \gamma_i^{\mathrm{L}}, \gamma_i^{\mathrm{U}} - c_i(x^0)] + \epsilon^{\mathrm{CR}}, & i \in \mathcal{C}^{\mathrm{R}} \end{cases} \tag{2.12}$$

and

$$t_i^0 = \begin{cases} \max[0, x_i^0 - x_i^{\mathrm{L}}] + \epsilon^{\mathrm{BL}} & i \in \mathcal{B}^{\mathrm{L}}, \\ \max[0, x_i^{\mathrm{U}} - x_i^0] + \epsilon^{\mathrm{BL}} & i \in \mathcal{B}^{\mathrm{U}}, \\ \max[0, x_i^0 - x_i^{\mathrm{L}}, x_i^{\mathrm{U}} - x_i^0] + \epsilon^{\mathrm{BR}}, & i \in \mathcal{B}^{\mathrm{R}}, \end{cases} \tag{2.13}$$

where ϵ^{E}, ϵ^{CL}, ϵ^{CU}, ϵ^{CR}, ϵ^{BL}, ϵ^{BU} and ϵ^{BR} are small positive constants. In practice, we might choose these constants all equal, e.g. to 0.1, or different to push (x^0, s^0, t^0) away from some constraint boundaries.

The starting point is initialized when the class is instantiated. The initialization chooses x_0 as specified in the model. Listing 4 sets s_0 and t_0 as in (2.12) and (2.13).

Listing 4. Elastic initialization

```
        # Constant which determines initial elastics: s = max(0,-c) + ethresh
        self.ethresh = 1.1
        self.tiny = 1.0e-8

        # Set initial elastics so (x0, s0) is strictly feasible
        # Note that a single elastic suffices for a range constraint
        self.s = numpy.zeros( self.nConst, 'd' )
        (self.c, self.cR) = self.evalC( self.x0 )
        for i in self.Eq + self.Ineq:
            self.s[i] = max( 0.0, -self.c[i] ) + self.ethresh
        for i in self.Range:
            self.s[i] = max( 0.0, - self.cR[i],
                             - self.c[i], ) + self.ethresh

        # Set initial elastics for the bound constraints so (x0, t0)
        # strictly satisfies the bounds. Single elastic for 2-sided bounds.

        self.t = {}
        for i in self.lowerB:
            self.t[i] = max(0.0, self.Lvar[i] - self.x0[i]) + self.ethresh
        for i in  self.upperB:
```

Given the form (2.11) of the barrier-penalty objective function, it is (tedious and lenghty but) straightforward to compute its first and second derivatives. We now expand them. We first define the primal Lagrange multiplier estimates

$$\begin{aligned} y_i^{\mathrm{E}} &= \mu_i(c_i(x) - \gamma_i^{\mathrm{E}} + s_i)^{-1}, & i \in \mathcal{E} \\ y_i^{\mathrm{L}} &= \mu_i(c_i(x) - \gamma_i^{\mathrm{L}} + s_i)^{-1}, & i \in \mathcal{C}^{\mathrm{L}} \\ y_i^{\mathrm{U}} &= \mu_i(\gamma_i^{\mathrm{U}} - c_i(x) + s_i)^{-1}, & i \in \mathcal{C}^{\mathrm{U}} \\ y_i^{\mathrm{RL}} &= \mu_i(c_i(x) - \gamma_i^{\mathrm{L}} + s_i)^{-1}, & i \in \mathcal{C}^{\mathrm{R}} \\ y_i^{\mathrm{RU}} &= \mu_i(\gamma_i^{\mathrm{U}} - c_i(x) + s_i)^{-1}, & i \in \mathcal{C}^{\mathrm{R}} \end{aligned} \tag{2.14}$$

for the general constraints,

$$\begin{aligned} z_i^{\mathrm{L}} &= \mu_i(x_i - x_i^{\mathrm{L}} + t_i)^{-1}, & i \in \mathcal{B}^{\mathrm{L}} \\ z_i^{\mathrm{U}} &= \mu_i(x_i^{\mathrm{U}} - x_i + t_i)^{-1}, & i \in \mathcal{B}^{\mathrm{U}} \\ z_i^{\mathrm{RL}} &= \mu_i(x_i - x_i^{\mathrm{L}} + t_i)^{-1}, & i \in \mathcal{B}^{\mathrm{R}} \\ z_i^{\mathrm{RU}} &= \mu_i(x_i^{\mathrm{U}} - x_i + t_i)^{-1}, & i \in \mathcal{B}^{\mathrm{R}}, \end{aligned} \tag{2.15}$$

for the bound constraints,

$$u = S^{-1}\mu, \tag{2.16}$$

for the elastics s associated to the general constraints and finally

$$v = T^{-1}\mu, \tag{2.17}$$

for the elastics t associated to the bounds.

2.2. Derivatives of the merit function

The gradient of (2.11) is the $(n + |\mathcal{E}| + |\mathcal{C}| + |\mathcal{B}|) \times 1$ vector given by

$$\nabla_{xst}\phi^{\mathrm{B}}(x, s, t; \mu, \nu) = \begin{bmatrix} \nabla_x \phi^{\mathrm{B}}(x, s, t; \mu, \nu) \\ \nabla_s \phi^{\mathrm{B}}(x, s, t; \mu, \nu) \\ \nabla_t \phi^{\mathrm{B}}(x, s, t; \mu, \nu) \end{bmatrix}, \tag{2.18}$$

where

$$\begin{aligned} \nabla_x \phi^{\mathrm{B}}(x, s; \mu, \nu) = \nabla_x \phi^{\mathrm{P}}(x, s, t; \nu) &- \sum_{i \in \mathcal{E}} y_i^{\mathrm{E}} \nabla c_i(x) - \sum_{i \in \mathcal{C}^{\mathrm{L}}} y_i^{\mathrm{L}} \nabla c_i(x) \\ &+ \sum_{i \in \mathcal{C}^{\mathrm{U}}} y_i^{\mathrm{U}} \nabla c_i(x) - \sum_{i \in \mathcal{C}^{\mathrm{R}}} [y_i^{\mathrm{RL}} - y_i^{\mathrm{RU}}] \nabla c_i(x) \\ &- \sum_{i \in \mathcal{B}^{\mathrm{L}}} z_i^{\mathrm{L}} e_i + \sum_{i \in \mathcal{B}^{\mathrm{U}}} z_i^{\mathrm{U}} e_i - \sum_{i \in \mathcal{B}^{\mathrm{R}}} [z_i^{\mathrm{RL}} - z_i^{\mathrm{RU}}] e_i, \end{aligned}$$

which we can rewrite, using (1.6), as

$$\nabla_x \phi^{\mathrm{B}}(x, s; \mu, \nu) = \nabla_x f(x) - J(x)^T \lambda(y, \nu) - \zeta(z),$$

where $\lambda(x, \nu) \in \mathbb{R}^{n_{\mathcal{E}} + n_{\mathcal{I}}}$ and $\zeta(z) \in \mathbb{R}^n$ are vectors such that

$$\lambda_i(y, \nu) = \begin{cases} y_i^{\mathrm{E}} - \nu_i, & i \in \mathcal{E} \\ y_i^{\mathrm{L}}, & i \in \mathcal{C}^{\mathrm{L}} \\ -y_i^{\mathrm{U}}, & i \in \mathcal{C}^{\mathrm{U}} \\ y_i^{\mathrm{RL}} - y_i^{\mathrm{RU}}, & i \in \mathcal{C}^{\mathrm{R}}, \end{cases} \qquad \zeta_i(z) = \begin{cases} z_i^{\mathrm{L}}, & i \in \mathcal{B}^{\mathrm{L}} \\ -z_i^{\mathrm{U}}, & i \in \mathcal{B}^{\mathrm{U}} \\ z_i^{\mathrm{RL}} - z_i^{\mathrm{RU}}, & i \in \mathcal{B}^{\mathrm{R}} \\ 0 & \text{otherwise}, \end{cases} \tag{2.19}$$

and where $J(x)$ is the Jacobian matrix of the constraints $c_i(x)$ for all $i \in \mathcal{E} \cup \mathcal{C}^{\mathrm{L}} \cup \mathcal{C}^{\mathrm{U}} \cup \mathcal{C}^{\mathrm{R}}$. Similarly,

$$\nabla_s \phi^{\mathrm{B}}(x, s; \mu, \nu) = \nu - \sum_{i \in \mathcal{E}} (y_i^{\mathrm{E}} - \nu_i + u_i) e_i - \sum_{i \in \mathcal{C}^{\mathrm{L}}} (y_i^{\mathrm{L}} + u_i) e_i - \sum_{i \in \mathcal{C}^{\mathrm{U}}} (y_i^{\mathrm{U}} + u_i) e_i - \sum_{i \in \mathcal{C}^{\mathrm{R}}} (y_i^{\mathrm{RL}} + y_i^{\mathrm{RU}} + u_i) e_i,$$

and finally

$$\nabla_t \phi^{\mathrm{B}}(x, s; \mu, \nu) = \nu - \sum_{i \in \mathcal{B}^{\mathrm{L}}} (z_i^{\mathrm{L}} + v_i) e_i - \sum_{i \in \mathcal{B}^{\mathrm{U}}} (z_i^{\mathrm{U}} + v_i) e_i - \sum_{i \in \mathcal{B}^{\mathrm{R}}} (z_i^{\mathrm{RL}} + z_i^{\mathrm{RU}} + v_i) e_i.$$

The intent is to approximately minimize a quadratic model of (2.11) under no constraints in a trust region to find an approximate minimizer of (1.5) with fixed μ and ν. Once such an approximate minimizer has been found, μ and ν are updated and attention turns to the next round of minimization.

In this implementation, we propose to use the exact second derivatives of (2.11). As considerable numerical experience has shown, a model far superior to the primal model is given by using the *primal-dual* Hessian matrix, obtained by treating y, z, u and v as additional, independent, variables.

For brevity, we denote $w = (x, s, t, y, z, u, v)$. Upon defining suitable primal-dual estimates $\tilde{y}$, $\tilde{z}$, $\tilde{u}$ and $\tilde{v}$ to be used in place of the primal estimates (2.14)–(2.17), we have

$$H^{\text{PD}}(w; \mu, \nu) = \begin{bmatrix} H_{xx} & H_{xs} & H_{xt} \\ H_{xs}^T & H_{ss} & H_{st} \\ H_{xt}^T & H_{st}^T & H_{tt} \end{bmatrix},$$

where

$$H_{xx} = \nabla_{xx} L(x, \lambda(y, \nu)) + J(x)^T \Theta(x, \tilde{y}, s) J(x) + E^T \Xi(x, \tilde{z}, t) E$$

with

$$\theta_i(x, \tilde{y}, s) = \begin{cases} \dfrac{\tilde{y}_i^{\text{E}}}{c_i(x) - \gamma_i^{\text{E}} + s_i}, & i \in \mathcal{E} \\ \dfrac{\tilde{y}_i^{\text{L}}}{c_i(x) - \gamma_i^{\text{L}} + s_i}, & i \in \mathcal{C}^{\text{L}} \\ \dfrac{\tilde{y}_i^{\text{U}}}{\gamma_i^{\text{U}} - c_i(x) + s_i}, & i \in \mathcal{C}^{\text{U}} \\ \dfrac{\tilde{y}_i^{\text{RL}}}{c_i(x) - \gamma_i^{\text{L}} + s_i} + \dfrac{\tilde{y}_i^{\text{RU}}}{\gamma_i^{\text{U}} - c_i(x) + s_i}, & i \in \mathcal{C}^{\text{R}}, \end{cases}$$

and

$$\xi_i(x, \tilde{z}, t) = \begin{cases} \dfrac{\tilde{z}_i^{\text{L}}}{x_i - x_i^{\text{L}} + t_i}, & i \in \mathcal{B}^{\text{L}} \\ \dfrac{\tilde{z}_i^{\text{U}}}{x_i^{\text{U}} - x_i + t_i}, & i \in \mathcal{B}^{\text{U}} \\ \dfrac{\tilde{z}_i^{\text{RL}}}{x_i - x_i^{\text{L}} + t_i} + \dfrac{\tilde{z}_i^{\text{RU}}}{x_i^{\text{U}} - x_i + t_i}, & i \in \mathcal{B}^{\text{R}}, \end{cases}$$

and where $\nabla_{xx} L(x, \lambda)$ is the Hessian of the Lagrangian of (1.1) and E is the matrix composed of the rows of the $n \times n$ identity matrix corresponding to indices in $\mathcal{B}^{\text{L}} \cup \mathcal{B}^{\text{U}} \cup \mathcal{B}^{\text{R}}$. This last matrix $E^T \Xi(x, \tilde{z}, t) E$ is diagonal, the i-th element on the diagonal being $\xi_i(x, \tilde{z}, t)$ if $i \in \mathcal{B}^{\text{L}} \cup \mathcal{B}^{\text{U}} \cup \mathcal{B}^{\text{R}}$ and 0 otherwise.

Similarly,

$$H_{xs} = J(x)^T \widehat{\Theta}(x, \tilde{y}, s),$$

where $\widehat{\Theta}(x, y, s)$ differs from $\Theta(x, y, s)$ in the sign of the multipliers corresponding to upper bounds;

$$\hat{\theta}_i(x, \tilde{y}, s) = \begin{cases} \dfrac{\tilde{y}_i^{\text{E}}}{c_i(x) - \gamma_i^{\text{E}} + s_i}, & i \in \mathcal{E} \\ \dfrac{\tilde{y}_i^{\text{L}}}{c_i(x) - \gamma_i^{\text{L}} + s_i}, & i \in \mathcal{C}^{\text{L}} \\ \dfrac{-\tilde{y}_i^{\text{U}}}{\gamma_i^{\text{U}} - c_i(x) + s_i}, & i \in \mathcal{C}^{\text{U}} \\ \dfrac{\tilde{y}_i^{\text{RL}}}{c_i(x) - \gamma_i^{\text{L}} + s_i} - \dfrac{\tilde{y}_i^{\text{RU}}}{\gamma_i^{\text{U}} - c_i(x) + s_i}, & i \in \mathcal{C}^{\text{R}}, \end{cases}$$

and

$$H_{xt} = E^T \widehat{\Xi}(x, \tilde{z}, t),$$

$$H_{ss} = \tilde{U} S^{-1} + \Theta(x, \tilde{y}, s),$$

$$H_{st} = 0,$$

$$H_{tt} = \tilde{V} T^{-1} + \Xi(x, \tilde{z}, t),$$

where $\widehat{\Xi}$ is to Ξ as $\widehat{\Theta}$ is to Θ:

$$\hat{\xi}_i(x,\tilde{z},t) = \begin{cases} \dfrac{\tilde{z}_i^{\mathrm{L}}}{x_i - x_i^{\mathrm{L}} + t_i}, & i \in \mathcal{B}^{\mathrm{L}} \\ \dfrac{-\tilde{z}_i^{\mathrm{U}}}{x_i^{\mathrm{U}} - x_i + t_i}, & i \in \mathcal{B}^{\mathrm{U}} \\ \dfrac{\tilde{z}_i^{\mathrm{RL}}}{x_i - x_i^{\mathrm{L}} + t_i} - \dfrac{\tilde{z}_i^{\mathrm{RU}}}{x_i^{\mathrm{U}} - x_i + t_i}, & i \in \mathcal{B}^{\mathrm{R}}. \end{cases}$$

To summarize, the primal-dual Hessian matrix is given by (dropping most arguments)

$$H^{\mathrm{PD}}(w;\mu,\nu) = \begin{bmatrix} \nabla_{xx}L(x,\lambda) + J^T\Theta J + E^T\Xi E & J^T\widehat{\Theta} & E^T\widehat{\Xi} \\ \widehat{\Theta}J & \tilde{U}S^{-1} + \Theta & 0 \\ \widehat{\Xi}E & 0 & \tilde{V}T^{-1} + \Xi \end{bmatrix}. \tag{2.20}$$

Note that the Hessian of the Lagrangian appearing in the (1,1) block is evaluated at x and the *shifted* multipliers $\lambda(y,\nu)$ (2.19).

3. The inner iteration

3.1. Trust-region subproblems

For fixed positive values of μ and ν, the inner iteration consists in solving

$$\begin{array}{ll} \underset{d}{\text{minimize}} & \nabla_{xst}\phi^{\mathrm{B}}(x,s,t;\mu,\nu)^T d + \frac{1}{2} d^T H^{\mathrm{PD}}(w;\mu,\nu) d \\ \text{subject to} & \|d\| \leq \Delta, \end{array} \tag{3.21}$$

using a traditional trust-region method. To this end, we must be able to compute matrix-vector products $H^{\mathrm{PD}}(w;\mu,\nu)d$ for some vector d, which requires us to be able to compute matrix-vector products with $\nabla_{xx}L(x,\lambda)$, $J(x)$ and $J(x)^T$, since all other matrices appearing in the blocks of (2.20) are diagonal.

3.2. Preconditioning the trust-region iteration

Using GLTR, we also need to solve, at each iteration, symmetric positive semi-definite preconditioning systems of the form

$$\begin{bmatrix} P + J^T\Theta J + \Xi & J^T\widehat{\Theta} & \Xi \\ \widehat{\Theta}J & \tilde{U}S^{-1} + \widehat{\Theta} & 0 \\ \Xi & 0 & \tilde{V}T^{-1} + \widehat{\Xi} \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_t \end{bmatrix} = \begin{bmatrix} r_x \\ r_s \\ r_t \end{bmatrix},$$

for appropriate right-hand sides r given by GLTR in the reverse communication phase, where $P \approx \nabla_{xx}L(x,\lambda(y,\nu))$. This may be done using, for instance, inertia-revealing factorizations such as `MA27` or `MA57` for the Harwell Subroutine Library [HSL00]. However, the matrix $J(x)^T\Theta J(x)$ might be dense, hampering a direct factorization.

3.3. Updating the dual variables

A step towards new candidate primal-dual multipliers may be obtained by considering the Newton step on the primal-dual system

$$\nabla_{xst}\phi^{\mathrm{B}}(x,s,t;\mu,\nu) = 0, \tag{3.22a}$$
$$(c_i(x) - \gamma_i^{\mathrm{E}} + s_i)y_i^{\mathrm{E}} - \mu_i = 0, \quad i \in \mathcal{E} \tag{3.22b}$$
$$(c_i(x) - \gamma_i^{\mathrm{L}} + s_i)y_i^{\mathrm{L}} - \mu_i = 0, \quad i \in \mathcal{C}^{\mathrm{L}} \tag{3.22c}$$
$$(\gamma_i^{\mathrm{U}} - c_i(x) + s_i)y_i^{\mathrm{U}} - \mu_i = 0, \quad i \in \mathcal{C}^{\mathrm{U}} \tag{3.22d}$$
$$(c_i(x) - \gamma_i^{\mathrm{L}} + s_i)y_i^{\mathrm{RL}} - \mu_i = 0, \quad i \in \mathcal{C}^{\mathrm{R}} \tag{3.22e}$$
$$(\gamma_i^{\mathrm{U}} - c_i(x) + s_i)y_i^{\mathrm{RU}} - \mu_i = 0, \quad i \in \mathcal{C}^{\mathrm{R}} \tag{3.22f}$$
$$Su - \mu = 0, \tag{3.22g}$$
$$Tv - \mu = 0, \tag{3.22h}$$
$$(x_i - x_i^{\mathrm{L}} + t_i)z_i^{\mathrm{L}} - \mu_i = 0, \quad i \in \mathcal{B}^{\mathrm{L}} \tag{3.22i}$$
$$(x_i^{\mathrm{U}} - x_i + t_i)z_i^{\mathrm{U}} - \mu_i = 0, \quad i \in \mathcal{B}^{\mathrm{U}} \tag{3.22j}$$
$$(x_i - x_i^{\mathrm{L}} + t_i)z_i^{\mathrm{RL}} - \mu_i = 0, \quad i \in \mathcal{B}^{\mathrm{R}} \tag{3.22k}$$
$$(x_i^{\mathrm{U}} - x_i + t_i)z_i^{\mathrm{RU}} - \mu_i = 0. \quad i \in \mathcal{B}^{\mathrm{R}} \tag{3.22l}$$

An approximate step (d_x, d_s, d_t) is obtained from the trust-region subproblem (3.21). The Newton equations from (3.22g)–(3.22h) then give

$$d_u = \mu S^{-1} - u - S^{-1}Ud_s, \tag{3.23a}$$
$$d_v = \mu T^{-1} - v - T^{-1}Vd_t. \tag{3.23b}$$

Similary, the steps in the multipliers y are obtained from (3.22b)–(3.22f)

$$dy_i^{\mathrm{E}} = \frac{\mu_i}{c_i(x) - \gamma_i^{\mathrm{E}} + s_i} - y_i^{\mathrm{E}} - \theta_i\left([d_s]_i + \nabla c_i(x)^T d_x\right) \tag{3.24a}$$
$$dy_i^{\mathrm{L}} = \frac{\mu_i}{c_i(x) - \gamma_i^{\mathrm{L}} + s_i} - y_i^{\mathrm{L}} - \theta_i\left([d_s]_i + \nabla c_i(x)^T d_x\right) \tag{3.24b}$$
$$dy_i^{\mathrm{U}} = \frac{\mu_i}{\gamma_i^{\mathrm{U}} - c_i(x) + s_i} - y_i^{\mathrm{U}} - \theta_i\left([d_s]_i - \nabla c_i(x)^T d_x\right) \tag{3.24c}$$
$$dy_i^{\mathrm{RL}} = \frac{\mu_i}{c_i(x) - \gamma_i^{\mathrm{L}} + s_i} - y_i^{\mathrm{RL}} - \frac{y_i^{\mathrm{RL}}}{c_i(x) - \gamma_i^{\mathrm{L}} + s_i}\left([d_s]_i + \nabla c_i(x)^T d_x\right) \tag{3.24d}$$
$$dy_u^{\mathrm{RU}} = \frac{\mu_i}{\gamma_i^{\mathrm{U}} - c_i(x) + s_i} - y_i^{\mathrm{RU}} - \frac{y_i^{\mathrm{RU}}}{\gamma_i^{\mathrm{U}} - c_i(x) + s_i}\left([d_s]_i - \nabla c_i(x)^T d_x\right). \tag{3.24e}$$

and the steps in the multipliers z are obtained from (3.22i)–(3.22l)

$$dz_i^{\mathrm{L}} = \frac{\mu_i}{x_i - x_i^{\mathrm{L}} + t_i} - z_i^{\mathrm{L}} - \xi_i\left([d_t]_i + [d_x]_i\right) \tag{3.25a}$$
$$dz_i^{\mathrm{U}} = \frac{\mu_i}{x_i^{\mathrm{U}} - x_i + t_i} - z_i^{\mathrm{U}} - \xi_i\left([d_t]_i - [d_x]_i\right) \tag{3.25b}$$
$$dz_i^{\mathrm{RL}} = \frac{\mu_i}{x_i - x_i^{\mathrm{L}} + t_i} - z_i^{\mathrm{RL}} - \frac{z_i^{\mathrm{RL}}}{x_i - x_i^{\mathrm{L}} + t_i}\left([d_t]_i + [d_x]_i\right) \tag{3.25c}$$
$$dz_i^{\mathrm{RU}} = \frac{\mu_i}{x_i^{\mathrm{U}} - x_i + t_i} - z_i^{\mathrm{RU}} - \frac{z_i^{\mathrm{RU}}}{x_i^{\mathrm{U}} - x_i + t_i}\left([d_t]_i - [d_x]_i\right). \tag{3.25d}$$

Multipliers are then updated according to

$$y^+ = y + \alpha dy \tag{3.26a}$$
$$u^+ = u + \alpha du \tag{3.26b}$$
$$v^+ = v + \alpha dv \tag{3.26c}$$
$$z^+ = z + \alpha dz, \tag{3.26d}$$

where $\alpha \in (0, 1]$ is chosen to ensure nonnegativity of the new candidates, i.e.,

$$\alpha = \tau \min \left\{ \min_{dy_i<0} \frac{-y_i}{dy_i}, \min_{du_i<0} \frac{-u_i}{du_i}, \min_{dv_i<0} \frac{-v_i}{dv_i}, \min_{dz_i<0} \frac{-z_i}{dz_i} \right\}, \tag{3.27}$$

where $\tau \simeq 1$. We denote $(y^+_{\text{PD}}, z^+_{\text{PD}}, u^+_{\text{PD}}, v^+_{\text{PD}})$ the resulting primal-dual multipliers.

To safeguard the values of the primal-dual multipliers at a new trial point (x^+, s^+, t^+) while at the same time allowing superlinear asymptotic convergence, the candidate multipliers $(y^+_{\text{PD}}, z^+_{\text{PD}}, u^+_{\text{PD}}, v^+_{\text{PD}})$ are projected into the interval

$$\left[\begin{bmatrix} \overline{y}_{\text{L}} \\ \overline{z}_{\text{L}} \\ \overline{u}_{\text{L}} \\ \overline{v}_{\text{L}} \end{bmatrix}, \begin{bmatrix} \overline{y}_{\text{U}} \\ \overline{z}_{\text{U}} \\ \overline{u}_{\text{U}} \\ \overline{v}_{\text{U}} \end{bmatrix} \right],$$

where

$$\begin{aligned} \overline{y}^{\text{X}}_{\text{L}} &= \kappa^{\text{L}} \min[e, y^{\text{X}}_{\text{PD}}, (y^{\text{X}}_{\text{P}})^+], && \text{X} = \text{E, L, U, RL, RU} \\ \overline{y}^{\text{X}}_{\text{U}} &= \max[\kappa^{\text{U}} e, y^{\text{X}}_{\text{PD}}, \kappa^{\text{U}} \mu^{-1} e, \kappa^{\text{U}} (y^{\text{X}}_{\text{P}})^+], && \text{X} = \text{E, L, U, RL, RU} \end{aligned}$$

4. A Python class for SUPERB

4.1. Instantiation

4.2. Methods

References

FGK02. R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press / Brooks/Cole Publishing Company, second edition, 2002.

GOT03a. N. I. M. Gould, D. Orban, and Ph. L. Toint. An interior-point ℓ_1-penalty method for nonlinear optimization. Technical Report RAL-TR-2003-022, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2003.

GOT03b. N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr and SifDec, a Constrained and Unconstrained Testing Environment, revisited. *Transactions of the ACM on Mathematical Software*, 29(4):373–394, December 2003.

HSL00. Harwell Subroutine Library. *A collection of Fortran codes for large-scale scientific computation*. AERE Harwell Laboratory, http://www.numerical.rl.ac.uk/hsl, 2000.