University of Liège
Electrical Engineering and Computer Science Department
Sart Tilman B-28
4000 Liege, BELGIUM

Interior Point Methods: A Survey, Short Survey of Applications to Power Systems, and Research Opportunities

- Technical Report -

Mevludin Glavic, Louis Wehenkel

February, 2004.

CONTENTS:

1. Introduction

In the past fifteen years, research on Interior Point Methods (IPM) and their applications were very extensive. Both IPM theory and computational implementations evolved very fast. IPM variants are being extended to solve all kind of programs: from linear to nonlinear and from convex to non-convex, and they are being applied to solve many practical problems, including engineering optimization problems. The first known IPM method is Frisch's (1955) [1] logarithmic barrier method that was later extensively studied by Fiacco and McCormick [2]. The modern era of IPM started with Karmarkar's paper [3] and his IPM for Linear Programming (LP) where solution time up to 50 times faster than simplex method were reported. The name "interior point" comes from LP notation. Namely, IPM methods move through the interior of the feasible region towards the optimal solution. This is in contrast to the simplex algorithm, which follows a sequence of adjacent extreme points to the optimal solution. Because of this strong link to the LP almost every tutorial survey of IPM methods starts with IPM for LP and then extends it to the non-linear programming problems. This is the case with this report. IPM are usually classified into three main categories: Projective methods, affine-scaling methods, and primal-dual methods. Among the different IPMs the primal-dual (including primal-dual algorithms that incorporate predictor and corrector) algorithms have gained a reputation for being the most efficient. The main steps in every IPM method are [4]: transforming an inequality constrained optimization problem to equality constrained one, formulate Lagrange function using logarithmic barrier functions, set the first-order optimality conditions, and apply Newton's method to the set of equations coming from the first-order optimality conditions. This report is organized in such a way to reflect these main steps. For the first the equality constrained nonlinear optimization problem is considered. Since the main breakthrough in IPM methods has been made in Karmarkar's paper [3] within the context of linear programming, the LP is then considered. The approach is then extended to nonlinear problems. A survey of IPM methods applications in solving different electric power system optimization problems is given. Some research opportunities are identified and shortly discussed in the report. The review of available software computational engines is given. Finally, a small NLP example is included in the paper and a NLP solver, built around SPOOLES (SParse Object-Oriented Linear Equations Solver) [5] as a linear algebra kernel, is applied to solve the problem.

2. Interior Point Methods: A tutorial survey [6,7]

2.1. Equality constrained nonlinear optimization problem

An equality constrained nonlinear optimization problem has the form,

Minimize $F(x)$
Subject to $g(x) = 0$ (1)

Where $F : R^n \to R$ and $g : R^n \to R^m$ are general smooth functions. The optimality conditions to (1) can be formulated using Lagrange function,

$$L(x, y) = F(x) - y^T g(x) \qquad (2)$$

y are known as Lagrange multipliers. Now, the first order optimality conditions to (1) are,

$$\begin{aligned} \nabla_x L(x, y) = \nabla F(x) - \nabla g(x)^T y = 0 \\ \nabla_y L(x, y) = -g(x) = 0 \end{aligned} \qquad (3)$$

The optimality conditions (3) are in general only necessary for optimality. Hence, an optimal solution to (1) must satisfy (3), but a solution to (3) is not necessary an optimal solution to (1). If the objective function is convex and the constraints are convex, then the first order optimality conditions (3) are also sufficient.
In summary, in this section it has been shown how a nonlinear programming problem can be reduced to a set of nonlinear equations. Therefore, a solution method for the nonlinear programming problem (1) is to compute a solution to the first order optimality conditions. Thus, a method for solution of nonlinear equations is needed.

2.1.1. Newton's method

The method of choice, in most applications, for solving system of nonlinear equations is Newton's method. Assume $f : R^n \to R^n$ is general smooth nonlinear function and a solution to the system

$$f(x) = 0, \qquad (4)$$

is required. It follows from Taylor's theorem that,

$$f(x^0 + \Delta_x) \approx f(x^0) + \nabla f(x^0)\Delta_x, \qquad (5)$$

where $x^0, \Delta_x \in R^n$. If x^0 is an initial guess for the solution to (4), then Δ_x is to be computed such that $f(x^0 + \Delta_x) = 0$. In general this is impossible, but in view of (5) an approximation can be obtained from,

$$f(x^0) + \nabla f(x^0)\Delta_x = 0. \qquad (6)$$

The system (6) defines a set of linear equations in the variables Δ_x, which can be easily solved. Indeed, it is assumed that $\nabla f(x^0)$ is nonsingular, then Δ_x is given by,

$$\Delta_x = -\nabla f(x^0)^{-1} f(x^0).$$

Δ_x defines search direction and a new point x^1 is obtained from,

$$x^1 = x^0 + \alpha \cdot \Delta_x,$$

where α is a step size scalar. The plain Newton's method chooses $\alpha = 1$. Unfortunately, this does not secure convergence and therefore α has to be chosen in the interval $(0,1]$. One possible choice of α is given by,

$$\alpha^* = \arg\min_{\alpha \in (0,1]} \left\| f(x^0 + \alpha \cdot \Delta_x \right\|. \qquad (7)$$

Clearly this leads to an iterative method for solution of a set of nonlinear equations, which is terminated when $\|f(x)\| \approx 0$. The Newton's method is known to have good local convergence properties, i.e. the initial guess is close to the solution.
This method can be applied to the first order optimality conditions (3). Let (x^0, y^0) be an initial guess for the solution. Then the Newton search direction to the first optimality conditions is defined by,

$$\begin{bmatrix} \nabla^2 F(x^0) - \sum_{i=1}^{m} y_i \nabla^2 g_i(x^0) & -\nabla g_i(x^0)^T \\ -\nabla g_i(x^0) & 0 \end{bmatrix} \times \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} = -\begin{bmatrix} \nabla F(x^0) - \nabla g(x^0)^T y^0 \\ -g(x^0) \end{bmatrix} \qquad (8)$$

The new point is given by,

$$\begin{bmatrix} x^1 \\ y^1 \end{bmatrix} = \begin{bmatrix} x^0 \\ y^0 \end{bmatrix} + \alpha \cdot \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \qquad (9)$$

for a suitable chosen step size α. In general, Newton's method is considered as very powerful for nonlinear programming problems, but should be carefully applied and special care is to be given to: the choice of efficient solver for linear equations, the choice of initial guesses. In addition, to be able to apply this powerful method for optimization problems the problem of handling inequality constraints is to be resolved.

2.2. Interior Point Methods for LP

This section is concerned with a method for handling inequalities within Newton's method in order to be able to apply it in solving primal or dual LP problems.

2.2.1. Linear Programming (LP) problem

An LP is defined as minimizing or maximizing a linear function subject to linear constraints. The standard for of LP problem is as follows,

$$\begin{aligned} &\text{Minimize } c^T x \\ &\text{Subject to } Ax = b, \\ &\qquad\qquad x \geq 0 \quad . \end{aligned} \tag{10}$$

where,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix},$$

and,

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}.$$

2.2.2. The primal approach

Assume that an LP problem is to be solved. One way to get rid of the inequalities is as follows,

$$\begin{aligned} &\text{Minimize } c^T x - \mu \sum_{i=1}^{n} \ln(x_i) \\ &\text{Subject to } Ax = b, \\ &\qquad\qquad x > 0 \quad . \end{aligned} \tag{11}$$

where μ is a positive parameter. Recall,

$$\lim_{x_i \to 0} \ln(x_i) = -\infty \tag{12}$$

Therefore, the logarithmic term in the objective function acts as a barrier which penalizes nonpositive solutions. This implies any optimal solution to (11) will satisfy the inequalities $x \geq 0$ strictly, because of minimization in (11). Using this observation, the $x > 0$ inequalities in (11) may be dropped and an equality constrained nonlinear optimization problem is obtained. An optimal solution to (11) for a sufficiently small μ is a good approximate solution to LP problem. This can be proved from the optimality conditions to (11). First define the Lagrange function,

$$L(x, y) = c^T x - \mu \sum_{i=1}^{n} \ln(x_i) - y^T (Ax - b),$$

where $y \in R^m$ are Lagrange multipliers corresponding to the equality constraints in (10). Now differentiation gives,

$$\frac{\partial L}{\partial x_i} = c_i - \mu \cdot x_i^{-1} - A_{:i} y \quad and \quad \frac{\partial L}{\partial y_j} = b_j - A_{j:} x$$

In vector notation,

$$\begin{aligned} \nabla_x L(x, y) &= c - \mu X^{-1} e - A^T y = 0 \\ \nabla_y L(x, y) &= b - Ax = 0 \quad , \quad x > 0 \end{aligned} \tag{13}$$

In the notation used, the $A_{:i}$ and $A_{j:}$ are the i-th column and the j-th row of A, respectively. Moreover, $e := (1,...,1)^T$ and

$$X := diag(x) := \begin{bmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_n \end{bmatrix} \tag{14}$$

First order optimality conditions (13) can be rewritten by introducing the vector $s = \mu X^{-1} e$ leading to,

$$\begin{aligned} c - s - A^T y &= 0, \\ b - Ax &= 0, \quad x > 0, \\ s &= \mu X^{-1} e \end{aligned} \tag{15}$$

If both sides of the last equality of (15) are multiplied by X and using a minor reordering the result is,

$$\begin{aligned} A^T y + s &= c, \\ Ax &= b, \quad x > 0, \\ Xs &= \mu \cdot e. \end{aligned} \tag{16}$$

The first set of equalities in (16) enforces dual feasibility, the second set enforces primal feasibility, and the last set is the complementarity condition perturbed by μ.
Let $(x(\mu), y(\mu), s(\mu))$ be a solution to (16) for some $\mu > 0$. Then $x(\mu)$ is primal feasible. Furthermore,

$$A^T y(\mu) + s(\mu) = c \quad and \quad s(\mu) = \mu X(\mu)^{-1} e > 0,$$

which shows $(y(\mu), s(\mu))$ is dual feasible. In other words $(x(\mu), y(\mu), s(\mu))$ is a primal-dual feasible pair. Therefore, the duality gap can be computed as follows,

$$c^T x(\mu) - b^T y(\mu) = x(\mu)s(\mu) = e^T X(\mu)s(\mu) = e^T (\mu \cdot e) = \mu \cdot e^T e = \mu \cdot n$$

Any solution that satisfies (16), and is optimal solution to (11), defines a primal-dual feasible pair.
An important question is whether the objective function in (11) is convex, because in this case the optimality conditions are sufficient. This question leads to study the barrier function,

$$B_{\mu(x)} := c^T x - \mu \sum_{j=1}^{n} \ln(x_j) = \sum_{j=1}^{n} (c_j x_j - \mu \ln(x_j)).$$

The function $\ln(x)$ is concave, which implies $-\mu \ln(x)$ is convex. Therefore, the barrier function is a positive sum of convex functions, which implies that the barrier function is convex.

2.2.3. Duality

An important question when LP problem is solved is: How optimality of the computed solution is verified? One way of proving optimality is to generate a lower bound on the objective value. By definition, a valid lower bound z must satisfy condition,

$$c^T x \geq z, \quad \forall x \in X.$$

If solution $x^* \in X$ has been generated such that,

$$c^T x^* = z,$$

then x^* is optimal solution due to the fact z is lower bound on the optimal objective value. A method to generate valid lower bounds is through the definition of so-called dual problem. To each LP problem (10), also called primal problem, there is corresponding dual problem in the form,

Maximize $b^T y$

Subject to $A^T y + s = c,$

$$s \geq 0. \qquad (17)$$

The pair (y,s) is a dual feasible solution if it satisfies the constraints in (17). Also, (x,y,s) is said to be primal and dual feasible if x and (y,s) is primal and dual feasible, respectively. Such a pair is called a primal-dual feasible pair.
The difference,

$$c^T x - b^T y \qquad (18)$$

is called duality gap and,

$$x^T s \quad , \qquad (19)$$

is called complementary gap. Some useful definitions in regard to primal-dual approach are:

Primal feasibility: $Ax^* = b, \quad x^* \geq 0$,
Dual feasibility: $A^T y^* + s^* = c, \quad s^* \geq 0$,
Optimality: $c^T x^* - b^T y^* = 0$.

Hence, in practice the optimality of a computed solution can always be verified by checking primal and dual feasibility and that the dual gap is zero.

2.2.4. A dual approach

The barrier term can be introduced to the dual problem as follows,

Maximize $b^T y + \mu \sum_{j=1}^{n} \ln(s_j)$
Subject to $A^T y + s = c,$
$s > 0.$

Let x denote the Lagrange multipliers then the Lagrange function is given by,

$$L(x,y,s) = b^T y + \mu \sum_{j=1}^{n} \ln(s_j) - x^T (A^T y + s - c). \qquad (20)$$

The optimality conditions are,

$$\begin{aligned} \nabla_x L(x,y,s) &= c - s - A^T y = 0, \quad s > 0, \\ \nabla_y L(x,y,s) &= b - Ax = 0, \qquad (21) \\ \nabla_s L(x,y,s) &= \mu \cdot S^{-1} e - x = 0. \end{aligned}$$

This equation is equivalent to,

$$\begin{aligned} A^T y + s &= c, \quad s > 0, \\ Ax &= b, \qquad\qquad (22) \\ Xs &= \mu \cdot e. \end{aligned}$$

These equations are essentially the same as in the primal case.

2.2.5. The primal-dual approach

In both, primal and dual, cases a set of first order optimality conditions to the barrier problem were obtained. Combining these two sets of optimality conditions gives,

$$\begin{aligned} Ax &= b, \quad x > 0, \\ A^T y + s &= c, \quad s > 0, \qquad\qquad (23) \\ Xs &= \mu \cdot e. \end{aligned}$$

These conditions are called the perturbed Karush-Kuhn-Tucker (KKT) conditions, because they are identical to the KKT conditions to original LP problem, except the complementary conditions have been perturbed by μ. Therefore, a solution to (23) for a sufficiently small μ is a good approximation to the optimal solution.
The system (23) defines a set of nonlinear equations which can be solved using Newton's method. This is exactly the main of the so-called primal-dual algorithm. Let define the nonlinear function,

$$F_\gamma((x, y, s) := \begin{bmatrix} Ax - b \\ A^T y + s - c \\ Xs - \gamma \cdot \mu \cdot e \end{bmatrix},$$

where $\mu := x^T s / n$ and $\gamma \geq 0$. Here, μ is not parameter and γ is introduced instead. In addition, μ is defined to be the average complementary product.
Assume $(\bar{x}, \bar{y}, \bar{s})$ is given such that $\bar{x} > 0$ and $\bar{s} > 0$, then one iteration of Newton's method applied to the system,

$$F_\gamma(x, y, s) = 0$$

is identical to,

$$\nabla F_\gamma(\bar{x}, \bar{y}, \bar{s}) \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_s \end{bmatrix} = -F_\gamma(\bar{x}, \bar{y}, \bar{s}).$$

Using the fact,

$$\nabla F_\gamma(\bar{x},\bar{y},\bar{s}) = \begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ \bar{S} & 0 & \bar{X} \end{bmatrix}$$

can be obtained,

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ \bar{S} & 0 & \bar{X} \end{bmatrix} \times \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_s \end{bmatrix} = \begin{bmatrix} \bar{r}_P \\ \bar{r}_D \\ -\bar{X}_{\bar{S}} + \gamma \cdot \bar{\mu} \cdot e \end{bmatrix}, \qquad (24)$$

where

$$\bar{r}_P := b - A\bar{x}, \qquad (25)$$

and

$$\bar{r}_D := c - A^T \bar{y} - \bar{s}. \qquad (26)$$

are the primal and dual residuals, respectively. If the residual vectors are zero, then the current point $(\bar{x},\bar{y},\bar{s})$ is primal and dual feasible, respectively.
The first step of the primal dual algorithm consists of solving (24), and then a new point,

$$\begin{bmatrix} x^k \\ y^k \\ s^k \end{bmatrix} = \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{s} \end{bmatrix} + \alpha \cdot \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_s \end{bmatrix}$$

is obtained for a suitable choice of α. The goal is to compute a solution such that the primal and dual residuals and the complementary gap $(x^T s)$ all are zero. The new primal residuals are given by,

$$r_P^k = b - Ax^k = b - A(\bar{x} + \alpha \cdot \Delta_x) = b - A\bar{x} - \alpha \cdot A\Delta_x = \bar{r}_P - \alpha \cdot \bar{r}_P = (1-\alpha) \cdot \bar{r}_P \qquad (27)$$

This shows that the new residuals are identical to the old residuals multiplied by the factor $(1-\alpha)$. If $\alpha \in (0,1]$, then the residuals are reduced. In particular, if $\alpha = 1$ then $r_P^k = 0$ showing the new point x^k satisfies the primal equality constraints exactly. In this respect the a large step size is beneficial. In similar way it can be shown that the dual residuals $(\bar{r}_D^k = (1-\alpha) \cdot \bar{r}_D)$ are also reduced. The new duality gap is identical to,

$$x^{k+1} s^k = (\bar{x} + \alpha \cdot \Delta_x)^T (\bar{s} + \alpha \cdot \Delta_s) = (1 - \alpha \cdot (1-\gamma)) \bar{x}^T \bar{s} + \alpha^2 \Delta_x^T \Delta_s \qquad (28)$$

For $\gamma \in (0,1]$ and a sufficiently small α the complementary gap is reduced. The smaller γ is the larger is the decrease in the gap. However, it should be noted that the search direction $(\Delta_x, \Delta_y, \Delta_s)$ is a function of γ and the step size α is implicitly a function of γ. In conclusion, by an appropriate choice of the step size the algorithm should converge.

2.2.6. Update of the variables

A large step size α implies a large reduction in the primal and dual residuals. Also, the reduction in the complementary gap tends to be proportional to the step size. However, the step size cannot be chosen arbitrarily large because the new point must satisfy the conditions $x^k > 0$ and $s^k > 0$. Therefore, the step size has to be strictly less than $\alpha^{\max}$ defined by,

$$\alpha^{\max} := \arg\max_{\alpha \geq 0} \left\{ \begin{bmatrix} \bar{x} \\ \bar{s} \end{bmatrix} + \alpha \cdot \begin{bmatrix} \Delta_x \\ \Delta_s \end{bmatrix} \geq 0 \right\}.$$

This value is the maximum possible step size until one of the primal or dual variables hits its lower bound exactly. Therefore, a possible choice of α is,

$$\alpha = \min(1, \theta \cdot \alpha^{\max}),$$

where $\theta \in (0,1)$.

This choice of the step size does not guarantee convergence, but it usually works well in practice. In practice $\alpha^{\max}$ is computed as follows. First let,

$$\alpha_P^{\max} = \min_j \left\{ -x_j / (\Delta_x)_j : (\Delta_x)_{j<0} \right\}$$

and

$$\alpha_D^{\max} = \min_j \left\{ -s_j / (\Delta_s)_j : (\Delta_s)_{j<0} \right\}$$

then

$$\alpha^{\max} = \min(\alpha_P^{\max}, \alpha_D^{\max}).$$

2.2.7. A termination criterion

The primal-dual algorithm is terminated if,

$$\begin{aligned} \|Ax - b\| &\leq \varepsilon_P, \\ \|A^T y + s - c\| &\leq \varepsilon_D, \\ x^T s &\leq \varepsilon_G. \end{aligned}$$

where $\varepsilon_P, \varepsilon_D$ and ε_G are small positive constants.

2.2.8. Comparison with the simplex algorithm

The success of primal-dual algorithm and its variants, as well as IPM in general, comes (or at least was triggered) from its superiority with respect to simplex algorithm. The primal-dual algorithm has a set of advantages with respect to simplex algorithm, but also disadvantages that fortunately can be handled within the algorithm. Some of the advantages and disadvantages are enumerated below.
Advantages of primal-dual algorithm:

- The algorithm does not has any problems with degeneracies and the number of iterations is not related to the number of vertices in the feasible region.
- For large LP problems the algorithm uses significantly fewer iterations than simplex algorithm.
- Most implementations of the algorithm usually solve a LP problem in less than 100 iterations even though the problem may contain millions of variables.

Disadvantages of primal-dual algorithm:

- The algorithm cannot detect a possible infeasible or unbounded status of the problem, and in some sense the primal-dual algorithm is not complete. Fortunately, this problem can be handled using homogenous model [4,7].
- Each iteration of the primal-dual algorithm is computationally much more expensive than one iteration of the simplex algorithm. However, the total work performed to solve a LP problem is a product of the number of iterations and the work performed in each iteration. For a large LP problem (say more than 100 variables) the primal-dual algorithms outperforms the simplex algorithm, and the bigger the problem size is this is the more pronounced.

2.3. Direct nonlinear IPM

A typical nonlinear programming problem can be mathematically expressed as,

Minimize $F(x)$
Subject to $g(x) = 0$

$$h_l \le h(x) \le h_u \qquad (29)$$

$$x_l \le x \le x_u$$

The same ideas, as in the case of LP problem, are applied. The first problem to solved is to transform the inequalities constrained problem to an equality constrained by introducing slack variables,

Minimize $F(x)$
Subject to $g(x) = 0$

$$
\begin{aligned}
&h(x)+s_h=h_u,\\
&s_h+s_{sh}=h_u-h_l,\\
&x+s_x=x_u,\\
&x-x_l\ge 0,\quad s_h,s_{sh},s_x\ge 0.
\end{aligned}
\qquad (30)
$$

The next step is to treat nonnegativity conditions in (30) by appending the logarithmic barrier functions to the objective function, and the resulting primal barrier problem is defined as follows,

Minimize $F_\mu=F(x)-\mu\sum_{i=1}^{n}\ln(x-x_l)_i-\mu\sum_{i=1}^{n}\ln(s_x)_i-\mu\sum_{j=1}^{m}\ln(s_h)_j-\mu\sum_{j=1}^{m}\ln(s_{sh})_j$

Subject to $g(x)=0$

$$
\begin{aligned}
&h(x)+s_h=h_u,\\
&s_h+s_{sh}=h_u-h_l,\\
&x+s_x=x_u.
\end{aligned}
\qquad (31)
$$

where the barrier parameter μ is a positive number. The Lagrangian function is,

$$
L_\mu=F(x)-y^Tg(x)-y_h^T[h_u-s_h-h(x)]-y_{sh}^T(h_u-h_l-s_h-s_{sh})-y_x^T(x_u-x-s_x)
$$
$$
-\mu\sum_{i=1}^{n}\ln(x-x_l)_i-\mu\sum_{i=1}^{n}\ln(s_x)_i-\mu\sum_{j=1}^{m}\ln(s_h)_j-\mu\sum_{j=1}^{m}\ln(s_{sh})_j
$$

(32)

where y, y_h, y_{sh} and y_x are Lagrangian multipliers. The first order optimality conditions are,

$$
\begin{aligned}
&\nabla_x L_\mu=\nabla F(x)-\nabla g(x)^T y+\nabla h(x)^T y_h+y_x-z=0,\\
&\nabla_{sh} L_\mu=y_h+y_{sh}-\mu\cdot S_h^{-1}e=0,\\
&\nabla_{ssh} L_\mu=y_{sh}-\mu\cdot S_{sh}^{-1}e=0,\\
&\nabla_{sx} L_\mu=y_x-\mu\cdot S_x^{-1}e=0,\\
&\nabla_y L_\mu=-g(x)=0,\\
&\nabla_{yh} L_\mu=h(x)+s_h-h_u=0,\\
&\nabla_{yx} L_\mu=x+s_x-x_u=0,\\
&\nabla_{ysh} L_\mu=s_h+s_{sh}-h_u+h_l=0,\\
&(X-X_l)\cdot Z\cdot e=\mu\cdot e
\end{aligned}
\qquad (33)
$$

where $e=[1,...,1]^T$, $X=diag(x_1,...,x_n)$, $S_h=diag(s_{h1},...,s_{hn})$, $Ssh=diag(s_{sh1},...,s_{shm})$, $X=diag(x_1,...,x_n)$, $Y_x=diag(y_{x1},...,y_{xm})$, $Y_{sh}=diag(y_{sh1},...,x_{shm})$, $Y_h=diag(y_{h1},...,y_{hn})$, and $S_x=diag(s_{x1},...,s_{xm})$.

The nonlinear equations (33) are then to be solved by some iterative method (Newton's or predictor-corrector) to obtain a search direction. The search direction is obtained from the next equations,

$$
\begin{bmatrix}
-Z^{-1}(X-X_l) & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 \\
0 & S_h^{-1}(Y_{sh}+Y_h) & 0 & 0 & 0 & I & I & 0 & 0 \\
0 & 0 & S_x^{-1}Y_x & 0 & I & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & S_{sh}^{-1}Y_{sh} & 0 & I & 0 & 0 & 0 \\
0 & 0 & I & 0 & 0 & 0 & 0 & I & 0 \\
0 & I & 0 & I & 0 & 0 & 0 & 0 & 0 \\
0 & I & 0 & 0 & 0 & 0 & 0 & -\nabla h & 0 \\
-I & 0 & 0 & 0 & I & 0 & -\nabla h^T & H & -\nabla g^T \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -\nabla g & 0
\end{bmatrix}
$$

$$
\times \begin{bmatrix} \Delta_z \\ \Delta_{sh} \\ \Delta_{sx} \\ \Delta_{ssh} \\ \Delta_{yx} \\ \Delta_{ysh} \\ \Delta_{yh} \\ \Delta_x \\ \Delta_y \end{bmatrix}
= \begin{bmatrix} -\mu \cdot Z^{-1}e + (X-X_l)\cdot e \\ \mu \cdot S_h^{-1}e - (Y_{sh}+Y_h)\cdot e \\ \mu \cdot S_x^{-1} - Y_x e \\ \mu \cdot S_{sh}^{-1} - Y_{sh}e \\ -x - s_x + x_u \\ -s_h - s_{sh} + (h_u - h_l) \\ -h(x) - s_h + h_u \\ -\nabla_x L_\mu \\ g(x) \end{bmatrix}
\qquad (34)
$$

Along the search direction, a step size α is chosen to preserve the non-negativity conditions. The new primal and dual variables are computed from,

$$
\begin{aligned}
x^k &= \bar{x} + \bar{\alpha}_P \Delta_x, \quad & y^k &= \bar{y} + \bar{\alpha}_D \Delta_y \\
s_x^k &= \bar{s}_x + \bar{\alpha}_P \Delta_{sx}, \quad & y_x^k &= \bar{y}_x + \alpha_D \Delta_{yx} \\
s_h^k &= \bar{s}_h + \bar{\alpha}_P \Delta_{sh}, \quad & y_h^k &= \bar{y}_h + \bar{\alpha}_D \Delta_{yh} \\
s_{sh}^k &= \bar{s}_{sh} + \bar{\alpha}_P \Delta_{ssh}, \quad & y_{sh}^k &= \bar{y}_{sh} + \bar{\alpha}_D \Delta_{ysh}
\end{aligned}
\qquad (35)
$$

The iteration procedures are terminated as the relative complementary gap and the mismatches of first order optimality conditions are sufficiently small,

$$
\frac{gap}{1+\|Dual\_Obj\|} \le \varepsilon_1, \qquad (36)
$$

$$
\|l\arg est\_mismatch\_of\_KKT\| \le \varepsilon_2 \qquad (37)
$$

where,

$$gap = (y_h + y_{sh})^T s_h + y_{sh}^T s_{sh} + y_x^T s_x + z^T (x - x_l),$$

$$Dual\_Obj = F(x) - y^T g(x) - y_h^T [h_u - h(x)] - y_{sh}^T (h_u - h_l) - y_x^T (x_u - x) - z^T (x - x_l)$$

The outline of the method is as the following,

1. Initialization. Choose a proper starting point such that the non-negativity conditions are satisfied.
2. Compute the barrier parameter μ.
3. Solve the system of equations (34).
4. Determine the step size α and update the solution.
5. Convergence test. If the solution meets the convergence criterion, optimal solution is found, otherwise go back to step 2.

2.3.1. The predictor-corrector primal-dual IPM

In this algorithm rather than applying the Newton's method to KKT conditions to generate correction terms to the current estimate, the new point is directly substituted into KKT, yielding,

$$\begin{aligned}
&H\Delta x - \nabla g(x)^T \Delta y + \nabla h(x)^T \Delta y_h + \Delta y_x - \Delta z = -\nabla_x L_\mu \\
&(Y_h + Y_{sh} + \Delta Y_h + \Delta Y_{sh})(S_h + \Delta S_h) \cdot e = \mu \cdot e \\
&(Y_{sh} + \Delta Y_{sh})(S_{sh} + \Delta S_{sh}) \cdot e = \mu \cdot e \\
&(Y_x + \Delta Y_x)(S_x + \Delta S_x) \cdot e = \mu \cdot e \\
&(X - X_l + \Delta X)(Z + \Delta Z) \cdot e = \mu \cdot e \qquad\qquad (38) \\
&\nabla g(x)\Delta x = -g(x) \\
&\nabla h(x)\Delta x + \Delta s_h = h_u - s_h - h(x) \\
&\Delta x + \Delta s_x = x_u - x - s_x \\
&\Delta s_h + \Delta s_{sh} = h_u - h_l - s_h - s_{sh}
\end{aligned}$$

The following symmetrical system is obtained,

$$\begin{bmatrix} -Z^{-1}(X-X_l) & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 \\ 0 & S_h^{-1}(Y_{sh}+Y_h) & 0 & 0 & 0 & I & I & 0 & 0 \\ 0 & 0 & S_x^{-1}Y_x & 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & S_{sh}^{-1}Y_{sh} & 0 & I & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & 0 & I & 0 \\ 0 & I & 0 & I & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & 0 & -\nabla h & 0 \\ -I & 0 & 0 & 0 & I & 0 & -\nabla h^T & H & -\nabla g^T \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\nabla g & 0 \end{bmatrix}$$

$$\times \begin{bmatrix} \Delta_z \\ \Delta_{sh} \\ \Delta_{sx} \\ \Delta_{ssh} \\ \Delta_{yx} \\ \Delta_{ysh} \\ \Delta_{yh} \\ \Delta_x \\ \Delta_y \end{bmatrix} = \begin{bmatrix} -\mu\cdot Z^{-1}e+(X-X_l)\cdot e+Z^{-1}\Delta X\Delta Z\cdot e \\ \mu\cdot S_h^{-1}e-(Y_{sh}+Y_h)\cdot e-S_h^{-1}\Delta S_h(\Delta Y_h+\Delta Y_{sh})\cdot e \\ \mu\cdot S_x^{-1}-Y_xe-S_x^{-1}\Delta S_x\Delta Y_xe \\ \mu\cdot S_{sh}^{-1}-Y_{sh}e-S_{sh}^{-1}\Delta S_{sh}\Delta Y_{sh}e \\ -x-s_x+x_u \\ -s_h-s_{sh}+(h_u-h_l) \\ -h(x)-s_h+h_u \\ -\nabla_x L_\mu \\ g(x) \end{bmatrix} \qquad (39)$$

The major difference between (34) and (39) is the presence of the nonlinear terms $\Delta X\Delta Ze, \Delta S_h(\Delta Y_h+\Delta Y_{sh})e, \Delta S_x\Delta Y_xe$ and $\Delta S_{sh}\Delta Y_{sh}$ in the right hand side of (39). These nonlinear terms cannot be determined in advance, therefore (39) only can be solved approximately. In order to estimate these nonlinear terms, Mehrotra [8] suggests first solving the defining equations for the primal-dual affine directions,

$$\begin{aligned} &H\Delta\tilde{x}-\nabla g(x)^T\Delta\tilde{y}+\nabla h(x)^T\Delta\tilde{y}_h+\Delta\tilde{y}_x-\Delta\tilde{z}=-\nabla_x L_\mu \\ &(Y_h+Y_{sh})\Delta\tilde{s}_h+S_h(\Delta\tilde{y}_h+\Delta\tilde{y}_{sh})=-(Y_h+Y_{sh})S_he \\ &Y_{sh}\Delta\tilde{s}_{sh}+S_{sh}\Delta\tilde{y}_{sh}=-Y_{sh}S_{sh}e \\ &Y_x\Delta\tilde{s}_x+S_x\Delta\tilde{y}_x=-Y_xS_xe \\ &(X-X_l)\Delta\tilde{z}+Z\Delta\tilde{x}=-(X-X_l)Ze \qquad (40) \\ &\nabla g(x)\Delta\tilde{x}=-g(x) \\ &\nabla h(x)\Delta\tilde{x}+\Delta\tilde{s}_h=h_u\tilde{s}_h-h(x) \\ &\Delta\tilde{x}+\Delta\tilde{s}_x=x_u-x-s_x \\ &\Delta\tilde{s}_h+\Delta\tilde{s}_{sh}=h_u-h_l-s_h-s_{sh} \end{aligned}$$

The values $\Delta\tilde{x},\Delta\tilde{y}_x,\Delta\tilde{s}_h,\Delta\tilde{s}_{sh},\Delta\tilde{z},\Delta\tilde{y}_h,\Delta\tilde{y}_{sh}$ are then used to approximate the nonlinear terms in the right hand side of (39), and to dynamically estimate μ. Once the estimates of nonlinear terms in the right-hand side of (39) and μ are determined, the actual new search directions ($\Delta x, \Delta y_x, \Delta s_h, \Delta s_{sh}, \Delta z, \Delta y_h, \Delta y_{sh}$) is obtained by (39). The outline of this algorithm can be stated as follows,

1. Initialization. Choose a proper starting point such that the non-negativity conditions are satisfied.
2. Solve the system of equations (40).
3. Compute the barrier parameter μ and the estimated nonlinear terms.
4. Solve the system of equations (39).
5. Determine the step size α and update the solution.
6. Convergence test. If the solution meets the convergence criterion, optimal solution is found, otherwise go back to step 2.

2.3.2. Sparsity techniques

In the described algorithms, the major computational effort is solving the large, sparse linear systems of the form,

$$\begin{bmatrix} D^{-2} & A^T \\ \Lambda & 0 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r \\ s \end{bmatrix} \qquad (41)$$

Therefore, it is essential to consider efficient methods for their solution, which can be either direct or iterative methods [4,9]. Usually, for IPM the linear systems (41) are solved using direct factorizations and most of the methods considered are based on Rothberg'a [10] factorization method or sparse Cholesky factorization [11], etc. The reason that direct methods are usually applied is in difficulties in choosing a good and computationally cheap preconditioning algorithm for iterative methods [6,7] and they are not competitive, in general, with direct ones.

2.3.3. Some implementation issues

The implementation issues considered in this section include: the adjustment of the barrier parameter, the determination of the Newton's step size, accuracy, and the choice of the starting point (initialization).

The adjustment of barrier parameter. Discussion follows largely that from [4,12]. Based on Fiaco and McCormick's theorem [2], the barrier parameter μ must approach zero as the iterations progress. The primal-dual method itself suggests how μ should be reduced from step to step. For LP problems [7], the value of μ is made proportional to the duality gap. The duality gap of a nonlinear problem is defined as,

$$gap = y^T g(x) + y^T [h_u - h(x)] + y_{sh}^T (h_u - h_l) + y_x^T (x_u - x) + z^T (x - x_l) \qquad (42)$$

The gap is positive if the primal and dual variables meet all the primal and dual constraints and is zero at the optimum point. However, due to the fact that the primal and dual variables are not feasible, the value of the gap may not be positive. Therefore, the complementary gap can be used to approximate duality gap,

$$gap^{*} = (y_h + y_{sh})^T s_h + y_{sh}^T s_{sh} + y_x^T s_x + z^T (x - x_l) \qquad (43)$$

and following the results of [12] it can be chosen,

$$\mu = \frac{gap^{*}}{4(n+m)^2} \qquad (44)$$

for the pure primal-dual IPM. For the predictor-corrector primal-dual IPM,

$$\mu = \left(\frac{ga\hat{p}}{gap^{*}} \right)^2 \left(\frac{ga\hat{p}}{2(n+m)^2} \right) \qquad (45)$$

where,

$$\begin{aligned} ga\hat{p} &= (y_{sh} + \hat{\alpha}\Delta\tilde{y}_{sh})^T (s_{sh} + \hat{\alpha}\Delta\tilde{s}_{sh}) + (y_x + \hat{\alpha}\Delta\tilde{y}_x)^T (s_x + \hat{\alpha}\Delta\tilde{s}_x) + \\ & [y_h + y_{sh} + \hat{\alpha}(\Delta\tilde{y}_h + \Delta\tilde{y}_{sh})]^T (s_h + \hat{\alpha}\Delta\tilde{s}_h) + (z + \hat{\alpha}\Delta\tilde{z})^T (x - x_l + \hat{\alpha}\Delta\tilde{x}) \end{aligned} \qquad (46)$$

$$\hat{\alpha} = \min \left\{ \begin{array}{l} -\frac{(x - x_l)_j}{\Delta\tilde{x}_j}, -\frac{(s_x)_j}{(\Delta\tilde{s}_x)_j}, -\frac{(s_{sh})_j}{(\Delta\tilde{s}_{sh})_j}, -\frac{(s_h)_j}{(\Delta\tilde{s}_h)_j}, -\frac{(y_x)_j}{(\Delta\tilde{y}_x)_j}, -\frac{(y_{sh})_j}{(\Delta\tilde{y}_{sh})_j}, \\ -\frac{(y_{sh} + y_h)_j}{(\Delta\tilde{y}_{sh} + \Delta\tilde{y}_h)_j}, -\frac{z_j}{\Delta z_j}, 1.0 \end{array} \right\} \qquad (47)$$

for those,

$$\Delta\tilde{x}_j < 0, (\Delta\tilde{s}_x)_j < 0, (\Delta\tilde{s}_h)_j < 0, (\Delta\tilde{s}_{sh})_j < 0, (\Delta\tilde{y}_x)_j < 0, (\Delta\tilde{y}_{sh})_j < 0,$$
$$(\Delta\tilde{y}_h + \Delta\tilde{y}_{sh})_j < 0, \tilde{z}_j < 0.$$

<u>The Newton's step size</u>, α, is determined as,

$$\alpha = \min\{0.9995\alpha^{*}, 1.0\} \qquad (48)$$

where,

$$\hat{\alpha} = \min \left\{ \begin{array}{l} -\frac{(x - x_l)_j}{\Delta x_j}, -\frac{(s_x)_j}{(\Delta s_x)_j}, -\frac{(s_{sh})_j}{(\Delta s_{sh})_j}, -\frac{(s_h)_j}{(\Delta s_h)_j}, -\frac{(y_x)_j}{(\Delta y_x)_j}, -\frac{(y_{sh})_j}{(\Delta y_{sh})_j}, \\ -\frac{(y_{sh} + y_h)_j}{(\Delta y_{sh} + \Delta y_h)_j}, -\frac{z_j}{\Delta z_j} \end{array} \right\} \qquad (49)$$

for those,

$$\Delta x_j < 0, (\Delta s_x)_j < 0, (\Delta s_h)_j < 0, (\Delta s_{sh})_j < 0, (\Delta y_x)_j < 0, (\Delta y_{sh})_j < 0,$$
$$(\Delta y_h + \Delta y_{sh})_j < 0, \Delta z_j < 0.$$

The constant 0.9995 is chosen to prevent nonnegative variables from being zero. Because of this, the logarithmic barrier functions are continuous and differentiable.

The choice of starting point (initialization). Although a strictly feasible initial point is not mandatory for the algorithms, the initial point still needs to meet nonnegativity condition. One possibility in choosing initial point can be to choose initial state vector, x, as the middle point between the upper and lower bounds, while the initial values of the slack variables can be chosen arbitrary within their bounds. To determine the initial values of the dual variables Mehrotra's method [8] can be applied. According to that method, first it is necessary to define,

$$\xi = 1 + \left\| \nabla F(x^0) \right\| \qquad (50)$$

where $\|\bullet\|$ is the l_1 norm. Then for each $j = 1,...,n$

$$\begin{aligned} z_j &= \nabla F_j + \xi, \quad (y_x)_j = \xi, \quad if \quad \nabla F_j > \xi \\ z_j &= -\nabla F_j, \quad (y_x)_j = -2\nabla F_j, \quad if \quad \nabla F_j < -\xi \\ z_j &= \nabla F_j + \xi, \quad (y_x)_j = \xi, \quad if \quad 0 \le \nabla F_j < \xi \\ z_j &= \xi, \quad (y_x)_j = \nabla F_j + \xi, \quad if \quad -\xi \le \nabla F_j < 0 \end{aligned} \qquad (51)$$

and $y^0 = e, y_h^0 = 0, y_{sh}^0 = 1.5\xi \cdot e$.

3. Interior Point Codes: a survey

Answering the question "What algorithm to use?" when one is faced with the optimization problem in [13] Boyd and Vandenberghe provided a simple relation that can roughly answer the question. The relation is:

$$Time\_to\_answer = (code\_time + \#\,ins\tan ces \times run\_time)$$

Code_time is the time spent to code the algorithm. IPMs are well known as hard to code, debug, and maintain. *Instances* is the number of the problem instances to be solved. Certainly, if one is allowed to start from already available software for an IPM the time to code it is becoming virtually zero and good run-time performances of the algorithms becoming more pronounced.

3.1. The codes

Several public domain, optimized, software codes are currently available on the Internet [4,7,9,14,15]. Some restrictions for the free use of these packages may apply. Also, there is a number of commercial software packages [9,14]. Below is a listing of public domain IPM software codes, followed by a short description of their key features.

- PCx (http://www.mcs.anl.gov/home/otc/Library/PCx/): This is IPM solver for LP, developed by the researchers at Argonne National Laboratory. The software is based on Mehrotra's predictor-corrector algorithm. Other features of the code include: standard MPS and callable subroutine input, pre-solving, and sparse Cholesky factorization of the normal equations by using Ng and Peyton code [11] with multiple minimum-degree ordering code of Liu [16] to reduce fill ins. PCx is available for Linux, UNIX, Windows 95, and Windows NT operating systems and makes use of both C and FORTRAN programming languages.
- HOPDM (http://ecoluinfo.unige.ch/~logilab/software/hopdm.html): An IPM solver for LP based on Mehrotra's predictor-corrector algorithm with (if requested) Gondzio's multiple corrections technique [17]. The main features of the code include: standard MPS and callable subroutine input, pre-solving (if requested), Cholesky factorization of the normal equations, minimum-degree ordering heuristic to reduce fill ins, handling of dense columns with a Schur's complement approach. The language is FOTRAN. QHOPDM is a version of HOPDM to solve quadratic programming problems. The version of the code for NLP is also developed and the code is available in C and FORTRAN programming languages.
- BPMPD (http://www.sztaki.hu/~meszaros/bpmpd/): BPMPD is based on Mehrotra's predictor-corrector algorithm and Gondzio's multiple corrections technique. Other features of the code are: standard MPS input, advanced pre-solver to reduce problem size, to eliminate features that could lead to numerical difficulties, and to make the matrix sparser, advanced heuristic to make decision between the normal equations and the augmented system forms, scaling of the problem for better numerical properties, advanced symbolic ordering for the normal equations, left-looking supernodal Cholesky factorization with loop unrolling, and increased numerical robustness (iterative refinement, etc.). The language is FORTRAN.
- LIPSOL (http://www.caam.rice.edu/~zhang/lipsol/): This is a Matlab-based software and is also based on Mehrotra's predictor-corrector algorithm. Lipsol uses Matlab's sparse matrix data structure, and MEX external interface facility to take advantage of existing efficient FORTRAN codes. It calls the sparse Cholesky solver of Ng and Peyton and multiple minimum degree ordering code of Liu.

Some of commercially available software tools, that are referred in the literature [9,14], are: CPLEX (http://www.cplex.com), OSL (http://www.research.ibm.com/osl/), and Xpress-MP (http://www.dash.co.uk/).

3.2. IPM sites on the Internet

Many Internet sites are devoted to IPM with a wealth of information on IPM theory, algorithm implementations, available software codes, and on practical applications of IPM. This fact might be confusing for a new-comer in the field. Instead to enumerate some of the many, here we propose two sites as excellent starting point (but nit just as the starting point, because for many of those interested in the field the sites provide enough information).

- (http://www.mcs.anl.gov/home/otc/InteriorPoint). Contains pointers to people and places where work on IPM is done. An extensive bibliography of IPM is assessable.
- (http://plato.la.asu.edu/guide.html). Decision tree for Optimization Software. This web page includes links to all the IPM software (most of them are freeware).

4. Interior point methods applications in power systems: a short survey

IPM have proven computationally to be viable alternative for the solution of several power engineering optimization problems. A variety of IPM has been applied to a number of power system problems, including:

- State estimation [18],
- Optimal power flow in general [9,12,19,20],
- Hydro-thermal coordination [14,21],
- Voltage collapse and reliability evaluation [22,23],
- Multi-reservoir management [see 9], and
- Fuel planning [32, also see 9].

The solution procedures considered in the applications include IPM applied to:

- A sequence of LP problems [25],
- A sequence of quadratic programming problems [26], and
- Directly applied to NLP problems [12].

A variety of IPM has been considered:

- A dual affine-scaling method [see 9],
- A number of plain primal-dual logarithmic barrier variants for NLP [see 9], and
- A number of predictor-corrector primal-dual logarithmic barrier variants for LP [see 9] and NLP [see 9].

The outstanding performances of the applications described in [4,9,14] are responsible for the growing interest in IPM for solving large, nonlinear power system problems. Computational results based on power networks ranging in size from 9-2423 buses [14] and 1832 and 3467 buses [14] show that the number of iterations required by the primal-dual logarithmic barrier IPM is not very sensitive to problem size, and the method is numerically robust. Some highly nonlinear problems such as unsolvable power flow (minimum load shedding) [22] and maximum loadability [23] problems, have been solved successfully.

5. Some research opportunities

This section discusses some advancements in IPM that are not widely exploited in engineering community (at least, such research activities were not reported). The advancements include: “hot start”, “warm start” [27], and combining IPM with global optimization techniques. Just a few publications reported on consideration of the above advancements [21,28,29]. In [28] the "hot start" and "warm start" were considered while in [21] the possibility of combining IPM with genetic algorithms is reported. The "hot start" and "warm start" refers to the idea of using some of the previous solutions to the optimization problem when one is facing the problem of solving multiple instance (perturbed) of the same problem.

5.1. Hot start

This approach refers to the idea of using the optimal solution of the original problem as the starting point for solving perturbed instance of the problem. The difficult associated with hot start in IPM is that the optimal solution of the original problem satisfies the complementary condition for a convergence tolerance parameter ε, but unless it is optimal to the perturbed problem as well, it is either primal or dual infeasible, or both [27,28]. Therefore, when restarting an IPM from this solution, a small initial value of μ will be chosen because the solution is close to a neighborhood of a minimizer for the original problem, making the matrix system in (34) or (40) ill-conditioned. Thus, it is important to improve the condition number as well as the initial value of μ to use the solution of the original problem as the initial point to the perturbed one. Due to the fact that the large condition number and small initial value of μ are caused by some variables at bounds, it is necessary to move these variables away from the boundary. A shift strategy, proposed in [28], is as follows (the shift is imposed on those variables close to the boundary),

$$\begin{aligned}
& x_j \leftarrow x_j + k \quad if \quad (x - x_l)_j < k \\
& (s_x)_j \leftarrow (s_x)_j + k \quad if \quad (s_x)_j < k \\
& (s_h)_j \leftarrow (s_h)_j + k \quad if \quad (s_h)_j < k \\
& (s_{sh})_j \leftarrow (s_{sh})_j + k \quad if \quad (s_{sh})_j < k \\
& z_j \leftarrow z_j + k \quad if \quad z_j < k \\
& (y_x)_j \leftarrow (y_x)_j + k \quad if \quad (y_x)_j < k \\
& (y_{sh})_j \leftarrow (y_{sh})_j + k \quad if \quad (y_{sh})_j < k \\
& (y_{sh} + y_h)_j \leftarrow (y_{sh} + y_h)_j + k \quad if \quad (y_{sh} + y_h)_j < k
\end{aligned} \tag{52}$$

where k is suggested to be set to 0.001 [28]. The new shifted point then is used as the initial point for perturbed problem. It is found in [28] (for the optimal power flow problem) that with hot start the perturbed problem can be solved with much fewer iterations and CPU times than ordinary ("cold") start. The percentage of time savings is about 40-50%.

5.2. Warm start

The warm start strategy does not use the optimal solution of the original problem. Because the solution point near the boundary causes ill-conditioning problem, it is possible to define alternative starting point that is close to optimality, yet is sufficiently far from the boundary of the feasible region. Thus, one of the intermediate solutions of the original problem is chosen as the starting point for the perturbed problem. The most important question is “how to pick up such a point in optimization process?”. In [28] the relative complementary slackness is used as the index for choosing the warm start point. An intermediate solution that satisfies,

$$\frac{gap^*}{1+|d\_obj|} \leq \tau \qquad (53)$$

where τ is predefined threshold, is stored as the solution for warm start.

5.3. Combining IPM with global optimization methods

Many engineering optimization problems are nonlinear with both continuous and discrete variables and different linear and nonlinear, time invariant and time varying, and equality and inequality constraints, and are nonconvex. Global optimization methods such as Genetic Algorithms (GA), simulated annealing, tabu search etc. have been considered and applied to solve the problem. It is already pointed out that IPM methods have been also applied to solve hard engineering optimization problems. Experience in using GA has shown that these methods have a deficiency that comes from GA’s weakness in local search. On the other hand, although successful applications in solving large-scale problems have been reported, IPMs have limited capabilities particularly in handling discrete variables. The combination of these two optimization techniques, and in general combination of global and local optimizers, comes quite naturally.
This has been recognized and some research efforts were undertaken in solving power system optimization problems [21,24,29]. One possibility is to combine the two techniques in such a way that GA deals with integer and discrete variables while local optimizer (IPM) deals with continuous variables and constraints. This approach have been considered in [24] for solving optimal reactive power dispatch problem and in [21] for solving short-term hydro-thermal coordination problem. Some other possibilities for combining global and local solvers have been considered in [29]. Since all available research reports on this topic can be regarded as recent seems that there is some space for improvement with the aim to find the best strategy on how to combine GA and IPM. Also, the use of Benders decomposition [30,31] in combination with both GA and IPM could be a challenging research topic. An application of Benders decomposition in solving power system optimization problems has been considered in [31].

6. A nonlinear example

The main purpose of a small example considered in this section is to demonstrate how the solution to the problem can be obtained using a NLP IPM. The program developed around sparse object-oriented library SPOOLES is employed as the direct nonlinear solver. The computational effort of each IPM algorithm is dominated by the solution of large, sparse linear system. Therefore, the performance of any IPM code is highly dependent on the linear algebra kernel. SPOOLES kernel, to best of our knowledge is not widely used but the reason it is chosen to be considered here are cited (see Netlib web-site: www.netlib.org) excellent performance of this linear algebra kernel.

6.1. The problem statement

Consider the following NLP problem with quadratic objective and constraint functions,

$$\begin{aligned}
&\textit{Minimize} \quad x_1^2 + x_2^2 - 4x_1 - 8x_2 + 20 \\
&\textit{Subject to} \\
&\qquad x_1^2 + x_2^2 - 2x_1 - 2x_2 - 2 = 0 \\
&\qquad 1 \le x_1^2 + x_2^2 - 6x_1 - 2x_2 + 10 \le 4
\end{aligned}$$

The problem is graphically illustrated in Figure 1. The feasible region of this problem is the line joining the pairs of points (B,C) and (D,E).

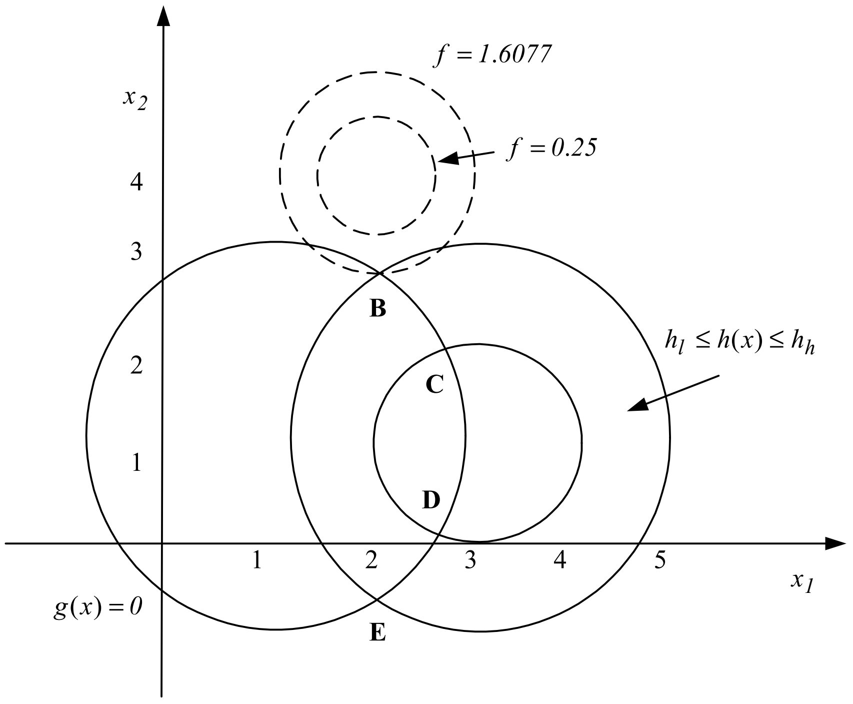

Figure 1. Nonlinear programming problem in two variables

6.2. Application of a direct NLP solver

For the particular problem considered we have,

$$\nabla_x f(x) = \begin{pmatrix} 2x_1 - 4 \\ 2x_2 - 8 \end{pmatrix}$$

$$J_g(x) = \begin{bmatrix} 2x_1 - 2 & 2x_2 - 2 \end{bmatrix}$$

$$J_h(x) = \begin{bmatrix} 2x_1 - 6 & 2x_2 - 2 \end{bmatrix}$$

where $\nabla_x f : R^n \to R^n$ is the gradient of $f(x)$ (a column vector), $J_g : R^n \to R^{m\times n}$ is the Jacobian of $g(x)$, and $J_h : R^n \to R^{p\times n}$ is the Jacobian of $h(x)$.
In addition to the Jacobians for direct solution the next information are also necessary,

$$\nabla_x^2 f(x) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix},$$
$$\nabla_x^2 g(x) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix},$$
$$\nabla_x^2 h(x) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}.$$

The procedure implemented in wrapper file follows that of section 2.3. The iteration procedure is terminated when:

$$\varepsilon_1 \le 10^{-5}$$
$$\varepsilon_2 \le 10^{-5}$$

The results of experimentation are summarized in Table I. The solution found in each considered case is

$$x_1 = 2.0000007$$
$$x_2 = 2.7320504$$

Initial point							Status	Positivity	No. of iterations
x1	x2	s	z	π	v	λ			
1.5	1.5	1.64	2.36	0.04444	0.08889	0	-*	+	8
5.0	4.0	1.64	2.36	0.04444	0.08889	0	-	+	7
2.75	0.032	1.64	2.36	0.04444	0.08889	0	+	+	9
2.75	1.968	1.64	2.36	0.04444	0.08889	0	+	+	6
1.5	1.5	2.25	0.75	0.04444	0.08889	0	-*	+	9
5.0	4.0	2.25	0.75	0.04444	0.08889	0	-	+	8
2.75	0.032	2.25	0.75	0.04444	0.08889	0	+	+	11
2.75	1.968	2.25	0.75	0.04444	0.08889	0	+	+	6

Table I. (Note: -* infeasible but satisfies inequality constraints, - infeasible, + feasible, for positivity: + satisfied, - not satisfied)

The feasibility of starting point is not necessary for the algorithm but some initialization heuristics is recommended (observe in presented results that the number of iterations is the smallest if point C indicated in Figure 1 is used as starting point). The heuristics depends on the practical problem considered. For recommended heuristics in solving optimal power flow problem see reference [9].

7. Conclusions

In this report an attempt has been made to present interior point methods for solving engineering optimization problems. The approach adopted is to put more emphasis on IPM essentials in such a way that the material be readable even for the readers not familiar with numeric optimization techniques. In addition, a short survey of the methods applications in solving power system optimization problems is included. A comprehensive list of relevant publications gives good starting points "what to read" and survey of freely available codes and the Internet sites devoted to the methods give good starting point for interested readers in future considerations (the fact is that there are a lot of publications and Internet sources, and for a new comer it is advantageous to have good starting points). A small nonlinear optimization problem is considered just for illustration purposes. A C++ wrapper for solving the problem)see the Appendix) has been developed around SPOOLES linear algebra kernel (SPOOLES library is freely available and can be downloaded from : http://www.netlib.org/linalg/spooles/spooles.2.2.html. The SPOOLES is just one of the available libraries. For those interested in developing own IPM software it is a suggestion to try some other available libraries (good starting point on the Internet is: http://www.netlib.org). References [33-41] are not referred throughout of the text but are provided here as useful material for further reading.

Bibliography:

[1] K. R. Frish: "The logarithmic potential method of conex programming, University Insitute of Economics, Oslo, Norway, Manuscipt, 1955.

[2] A. V. Fiacco, G. P. McCormick: "Nonlinear Programming: Sequential Unconstrained Minimization Techniques", John Willey & Sons, 1968.

[3] N. Karmarkar: "A new polynomial-time algorithm for linear programming", Combinatorica, Vol. 4, pp. 373-395, 1984.

[4] V. H. Quintana, G. L. Torres: "Introduction to Interior-Point Methods", IEEE PES task Force on Interior-Point Methods Applications to Power Systems, {Online} Available: http://wathvdc3.uwaterloo.ca/~iee-ipm, 1997.

[5] C. Ashcraft, R. Grimes, "*SPOOLES: An Object-Oriented Sparse Matrix Library*", Proceedings of the 1999 SIAM Conference on Parallel Processing for Scientific Computing, March 22-27, 1999.

[6] F. A. Potra, S. J. Wright: "Interior-Point Methods", Technical Report, [Online] Available: , 2000.

[7] S. J. Wright: "Primal-Dual Interior-Point Methods", SIAM, Philadelphia, 1996.

[8] S. Mehrotra: "On the implementation of a primal-dual interior point method", SIAM Journal on Optimization, Vol. 2, pp. 575-601, 1992.

[9] V. H. Quintana, G. L. Torres, J. Medina-Palomo: " Interior-Point Methods and Their Applications to Power Systems: A Classification of Publications and Software Codes", IEEE Transactions on Power Systems, Vol. 15, No. 1, pp. 170-175, 2000.

[10] E. Rothberg, A. Gupta: "Efficient sparse matrix factorization on high-performance workstations exploiting the memory hierarchy", ACM Transactions on Mathematical Software, Vol. 17, pp. 313-334, 1991.

[11] E. Ng, B. W. Peyton: “Block sparse Cholesky algorithms on advanced uniprocessor computers”, SIAM Journal on Scientific Computing, Vol. 14, pp. 1034-1056, 1993.

[12] Y. C. Wu, A. S. Debs, R. E. Marsten: “A Direct Nonlinear Predictor-Corrector Primal-Dual Interior Point Algorithm for Optimal Power Flows”, IEEE Transactions on Power Systems, Vol. 9, No. 2, pp. 876-883, 1994.

[13] S. Boyd, L. Vandenberghe: “Introduction to Convex Optimization with Engineering Applications”, Lecture Notes for EE392x, Electrical Engineering Department, Stanford University, 1995.

[14] J. Medina, V. H. Quintana, A. J. Conejo, F. P. Thoden: “A Comparison of Interior-Point Codes for Medium-Term Hydro-Thermal Coordination”, IEEE PES Summer Meeting, Paper 0-7803-1/97, 1997.

[15] L. S. Vargas, V. H. Quintana, A. Vanelli: “A Tutorial Description of an Interior Point Method and its Applications to Security-Constrained Economic Dispatch”, IEEE Transactions on Power Systems, Vol. 8, No. 3, pp. 1315-1324., 1993.

[16] M. Liu, S. K. Tso, Y. Cheng: “An Extended Nonlinear Primal-Dual Interior-Point Algorithm for Reactive Power Optimization of Large-Scale Power Systems with Discrete Control Variables”, IEEE Transactions on Power Systems, Vol. 17, No. 4, pp. 982-991, 2002.

[17] J. Gondzio: “Multiple centrality correctionsin a primal-dual method for linear programming”, Computational Optimization and Applications, Vol. 6, pp. 137-156, 1996.

[18] H. Singh, F. L. Alvarado: “Weighted least absolute value state estimation using interior point method”, IEEE Transactions on Power Systems, Vol. 9, pp. 1478-1484, 1994.

[19] R. A. Jabr, A. H. Cooninck, B. J. Cory: ”A Primal-Dual Interior Point Method for Optimal Power Flow Dispatching”, IEEE Transactions on Power Systems, Vol. 17, No. 3, pp. 654-662, 2002.

[20] J. Momoh, S. Guo, E. Ogbuobiri, R. Adapa: “The quadratic interior point method for solving power systems optimization problems”, IEEE Transactions on Power Systems, Vol. 9, pp. 1327-1336, 1994.

[21] J. L. M. Ramos, A. T. Lora, J. R. Santos, A. G. Exposito: “Short-Term Hydro-Thermal Coordination Based on Interior Point Nonlinear Programming and Genetic Algorithms”, IEEE Port PowerTech, Paper 0-7803-7139-9/01, Porto, Portugal, 2001.

[22] S. Grenville, J. C. O. Mello, A. C. G. Mello: “Application of interior point methods to power flow unsolvability”, IEEE Transactions on Power Systems, Vol. 11, pp. 1096-1103, 1996.

[23] G. D. Irrisari, X. Wang, J. Tong, S. Mokhtari: “Maximum loadability of power systems using interior point nonlinear optimization methods”, IEEE Transactions on Power Systems, Vol. 12, pp. 162-172, 1997.

[24] J. R. O. Soto, C. R. R. Dornellas, D. M. Falcao: “Optimal Reactive Power Dispatch using a Hybrid Formulation: Genetic Algorithms and Interior Point”, IEEE Porto PowerTech, Paper 0-7803-7139-9/01, Porto, Portugal, 2001.

[25] S. Greville: “Optimal reactive dispatch though interior point methods”, IEEE Transactions on Power Systems, Vol. 9, pp. 136-146, 1994.

[26] A.J. Urdaneta, J. F. Gomez, E. Sorrentino, L. Flores, R. Diaz: “A Hybrid Algorithm for Optimal Reactive Power Planning based upon Successive Linear

Programming", IEEE Transactions on Power Systems, Vol. 14, No. 4, pp. 1292-1298, 1999.

[27] J. Gondzio: "Warm start of the primal-dual method applied in the cutting plane scheme", Logilab, Technical Report 1994.3, University of Geneva, 1994.

[28] G. L. Torres, V. H. Quintana: "Rectangular Form Optimal Power Flow by Interior-Point Methods", In the Proceedings of COPIMERA'97, Santiago de Chile, pp. 64-70, 1997.

[29] K. Pannambalam, V. H. Quintana, A. Vanelli: "A Fast Algorithm for Power System Optimization Problems using an Interior Point Method", IEEE Transactions on Power Systems, Vol. 7, No. 2, pp. 892-899, 1992.

[30] J. N. Hooker, G. Ottoson: "Logic-Based Benders Decomposition", *Mathematical Programming* 96, pp. 33-60, 2003.

[31] N. Alguacil, A. J. Conejo: "Multiperiod Optimal Power Flow Using Benders Decomposition", IEEE Transactions on Power Systems, Vol. 15, No. 1, pp. 196-201, 2000.

[32] V. R. Sherkat, Y. Ikura: "Experience with Interior Point Optimization Software for a Fuel Planning Application" IEEE Transactions on Power Systems, Vol. 9, No. 2, 1994.

[33] C. H. Wu: "Hot-Sart and Decoupling in a Direct Nonlinear Interior Point Based Optimal Power Flows", IEEE Budapest PowerTech, Paper BPT99-260-16, Budapest, Hungary, 1999.

[34] H. Y. Benson, D. F. Shano, R. J. Wanderbei" A Comparative Study of Large-Scale Nonlinear Optimization Algorithms", Technical Report, ORFE-01-04, Princeton University, 2001.

[35] G. K. Purushothama, L. Jenkins: "Simulated Annealing With Local Search – A Hybrid Algorithm for Unit Commitment", IEEE Transactions on Power Systems, Vol. 18, No. 1, 2003.

[36] R. H. Byrd, M. E. Hribar, J. Nocedal: "An interior point algorithm for large scale nonlinear programming, Technical Report, University of Colorado, 1997.

[37] T. J. Carpenter, I. J. Lustig, J. M. Mulvey, D. F. Shanno: "Higuer-order predictor-corrector interior point methods with applications to quadratic objective", SIAM Journal on Optimization, Vol. 3., pp. 696-725, 1993.

[38] S. Wright: "Stability of augmented system factorizations in interior-point methods", Argonne national laboratory, Manuscipt, 1996.

[39] L. J. Lustig, R. E. Marsten, D. F. Shanno: "On implementing Mehrotra's predictor-corrector interior point method for linear programming", SIAM Journal on Optimization, Vol. 2, pp. 435-449, 1992.

[40] R. J. Vanderbei, D. F. Shanno: "An interior point algorithm for nonconvex nonlinear programming", Technical report, SOR-97-21, Princeton University, 1997.

[41] R. J. Vanderbei: "Interior Point Methods: Algorithms and Formulations", ORSA Journal on Computing, Vol. 6, pp. 32-34, 1994.

Appendix:

C++ wrapper file to solve example considered (this can be used as guide to implement solver for large-scale problems)

```
#include <math.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include "ipm_nlp.h"

int main ()
{
        bool radi = true;
        int row[18], col[18];
        double entr[18];
        Bridge *bridge;
        FILE *inpFile;
        memset(row, 0, 18* sizeof(int));
        memset(col, 0, 18* sizeof(int));
        memset(entr, 0, 18* sizeof(double));
        inpFile=fopen("input","r");

//      Read values of initial parameters and solution

        fscanf(inpFile,"  %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f\n",
                s,z,p,v,x1,x2,lambda);
        fclose(inpFile);

        Check validity

        if ((s < 0.) && (z<0.){
                fprintf(stdout,"Initial parameters are not valid")
                        return(0);
        }


        mi=0.1*(s*p+z*(p+v));
        int iterac = 0;
        df.initVec(2);
        Jg.initVec(2);
        Jh.initVec(2);

        dense1D b(7);

        bridge = Bridge_new() ;

        int permuteflag = 1 ;
        int error;


        Matrix A(7, 18, row, col, entr);

        bool setUp = false;
        int rc;
```

```
	int symmetryflag = SPOOLES_NONSYMMETRIC;
	dense1D dx(7);
	do
	{
		computeTests(A, b);
		A.goFirst();
		A.show(stdout);

		//For MATLAB if preferred
		//A.showMatlab(forMatlab, "Ab");
		//InpMtx_changeStorageMode(A.getMatrixPtr(),
INPMTX_BY_VECTORS);
		//A.show(stdout);
		//b.show(stdout);
		if (!setUp)
		{

//		Create Brisdge object

			Bridge_setMatrixParams(bridge,	7,	SPOOLES_REAL,
symmetryflag) ;

			if (bridge->symmetryflag != SPOOLES_NONSYMMETRIC)
				printf("Bridge is symetric!\n");

			bridge->pivotingflag = SPOOLES_PIVOTING;

			rc = Bridge_setup(bridge, A.getMatrixPtr()) ;
			setUp = true;

			if ( rc != 1 )
			{
				printf("Bridge Setup error...\n");
				return 0 ;
			}
		}

		rc = Bridge_factor(bridge, A.getMatrixPtr(), permuteflag, &error) ;

		if ( rc != 1 )
		{
			printf("Factorization error!!!\n");
			return 0;
		}

		rc	=	Bridge_solve(bridge,	permuteflag,	dx.getMatrixPtr(),
b.getMatrixPtr()) ;

		if ( rc != 1 )
```

```
{
        printf("Solving error..\n");
        return 0;
}
ro = s*p+z*(v+p);

mi = delta * ro / 2;

x1p = x1;
x2p = x2;

//Newton direction

alfa = 1;
if (dx[0] < 0)
        alfa = __min (-gama *s/dx[0], alfa);

if (dx[1] < 0)
        alfa = __min(-gama *z/dx[1], alfa);

if (dx[2] < 0)
        alfa = __min(-gama *p/dx[2], alfa);

if (dx[2]+dx[3] < 0)
{
        alfa = __min (-gama *(v+p)/(dx[2]+dx[3]), alfa);
}

s += alfa * dx[0];
z += alfa * dx[1];
p += alfa * dx[2];
v += alfa * dx[3];
x1 += alfa * dx[4];
x2 += alfa * dx[5];
lambda += alfa * dx[6];

ro = s*p+z*(v+p);

mi = 0.1 * ro;

e1 = v1();
e2 = v2();
e3 = v3();
e4 = v4();

//Initialization of vectors

df.setInit();
Jg.setInit();
Jh.setInit();
```

```
                    // Additional information can be printed here (if necessary)

                    //cout.width(8);
                    //cout.precision(4);
                    //cout << "Iter "<< iterac << " Alfa: "<< alfa << " Mi: " << mi << "
Tol: " << e1
                    //          <<" "<< e2 << " "<< e3 <<" "<< e4 << endl;
                    //cout.width(8);
                    //cout.precision(4);
                    //cout << "x= " << s <<" "<< z<<" " << p<<" " << v<<" " << x1<<" "
<< x2<<" " << lambda<< endl;

                    iterac=iterac+1;
//                  cout << "------------"<<endl<<"Solution found in " << iterac << "
iteration"<<endl;

                    if ((mi < epsmi) && (e1 < eps1) && (e2 < eps1) && (e3 < eps2) &&
(e4 < eps2))
                    {
//                            cout << "------------"<<endl<<"Solution found in " << iterac
<< " iteration"<<endl;
                              radi = false;
                    }
                    A.getMatrixPtr()->coordType   = INPMTX_BY_ROWS  ;
       A.getMatrixPtr()->storageMode = INPMTX_RAW_DATA ;
                    //A.show(stdout);
                    //A.showMatlab(forMatlab, "Af");
                    //b.show(stdout);

          } while (radi && iterac < 20);
          //cout.width(12);
          cout.precision(8);
          cout << endl<<"x1 =" << x1 << endl;
          cout << "x2 =" << x2 << endl;
          Bridge_free(bridge);
                    cout << "------------"<<endl<<"Solution found in " << iterac << "
iterations"<<endl;
          string str;
          cout << "IP finished! Type something and then <ENTER>" ;
          cin >> str;
          //fclose(forMatlab);

          return 0;
}
```

C++ header file to solve example considered (this can be used as guide to implement solver for large-scale problems)

```
extern "C"{
```

```
#include "../Bridge.h"
}
#define __max(a,b)  (((a) > (b)) ? (a) : (b))
#define __min(a,b)  (((a) < (b)) ? (a) : (b))
//using namespace std;

//class dense1D is one-dimensional vector for 1D real SPOOLES vector

class dense1D
{
	bool init;
	DenseMtx * m_dense;
	int m_dimension;
public:
	dense1D (int dimension):m_dimension(dimension), init(true)
	{
		m_dense = DenseMtx_new();
		DenseMtx_init(m_dense,  SPOOLES_REAL,  0,  0,  dimension,  1,  1,
dimension) ;

	};

	dense1D ():m_dimension(0),init(false)
	{
		m_dense = 0;
	};

	~dense1D ()
	{
		if (m_dense)
			DenseMtx_free(m_dense);
	};

	void initVec(int dimension)
	{
		m_dense = DenseMtx_new();
		DenseMtx_init(m_dense,  SPOOLES_REAL,  0,  0,  dimension,  1,  1,
dimension) ;
	};

	double &operator() (int i)
	{
		if (!init)
			DenseMtx_setRealEntry(m_dense, i, 0, 0.0) ;
		return m_dense->entries[i];
	};

	double &operator[] (int i)
	{
		if (!init)
```

```
                        DenseMtx_setRealEntry(m_dense, i, 0, 0.0) ;
                return m_dense->entries[i];
        };

        void setInit()
        {
                init = true;
        }

        DenseMtx * getMatrixPtr() {return m_dense;};

        double two_norm()
        {
                double norm;
                A2 a2;
                A2_setDefaultFields(&a2) ;
                DenseMtx_setA2(m_dense, &a2);
                norm = A2_twoNormOfColumn(&a2, 0) ;
                return norm;
        };

        void show(FILE *fp)
        {
                DenseMtx_writeForHumanEye(m_dense, fp) ;
        }


};

//class Matrix is OO wrapper for 2D real SPOOLES C matrix

class Matrix
{
private:
        bool init;
        InpMtx * mtx;
        double *m_pVal;
        DV *m_pEntries;
        IV *m_pRows;
        IV *m_pCols;
        int pos;

public:
        Matrix(int dimension, int nonZeros, int rows[], int cols[], double entries [] )
        :
          init(true),
                pos(0)
          {
                mtx = InpMtx_new() ;
                InpMtx_init(mtx, INPMTX_BY_ROWS, SPOOLES_REAL, 0, 0);
```

```
		m_pEntries =  &mtx->dvecDV;
		m_pEntries->vec = entries;
		m_pEntries->owned = 0;
		m_pEntries->size = m_pEntries->maxsize = nonZeros;

		m_pRows = &mtx->ivec1IV;
		m_pRows->owned = 0;
		m_pRows->size = m_pRows->maxsize =  nonZeros;
		m_pRows->vec = rows;

		m_pCols = &mtx->ivec2IV;
		m_pCols->owned = 0;
		m_pCols->size = m_pCols->maxsize =  nonZeros;
		m_pCols->vec = cols;

		mtx->maxnent = mtx->nent = nonZeros;
		mtx->resizeMultiple = 1.25;
		mtx->maxnvector = dimension;

	};

	Matrix (int dimension, int nonZeros)
		: init (false),
		pos(0)
	{
		mtx = InpMtx_new() ;
		InpMtx_init(mtx,	INPMTX_BY_ROWS,	SPOOLES_REAL,
nonZeros, dimension) ;
		m_pEntries =  &mtx->dvecDV;
		m_pRows = &mtx->ivec1IV;
		m_pCols = &mtx->ivec2IV;
	}

	Matrix (InpMtx * pMtx):	init(true),	mtx(pMtx), pos(0){};
	~Matrix ()
	{
		if (mtx)
			InpMtx_free(mtx);
	};

	void addEntry(int i, int j, double val)
	{
		InpMtx_inputRealEntry(mtx, i, j, val) ;
	}

	void show(FILE *fp)
	{
		InpMtx_writeForHumanEye(mtx, fp) ;
		printf("Size: %d nvector: %d\n", m_pEntries->size, mtx->nvector);
	};
```

```
        void showMatlab(FILE *fp, char *matrixName)
        {
                InpMtx_writeForMatlab(mtx, matrixName, fp) ;
        };

        double& operator() (int i, int j)
        {
                if (!init)
                {
                        InpMtx_inputRealEntry ( mtx, i, j, 0.0) ;
                        m_pVal= & (m_pEntries->vec[pos]);
                }
                else
                {
                        m_pVal= & (m_pEntries->vec[pos]);
                        m_pRows->vec[pos] = i;
                        m_pCols->vec[pos] = j;
                }
                pos++;
                return *m_pVal;
        };

        void setInit()
        {
                init = true;
        };

        void goFirst()
        {
                pos = 0;
        };

        InpMtx * getMatrixPtr()
        {
                return mtx;
        };
};

typedef double Type;

//Global variables

dense1D    df, Jg, Jh ;
double p,s,v,vk,z,x1,x2, x1p =0, x2p=0, lambda,mi, delta = 0.2, ro, gama=0.9995,
alfap, alfad, alfa;
double e1, e2, e3, e4;
double eps1 = 1e-5;
double eps2 = 1e-5;
double epsmi = 1e-5;
double hmin = 1;
double hmax = 4;

void computeTests(Matrix& A, dense1D& b)
{
        A(0,0) = p;
        A(0,2) = s;
        A(1,1) = v+p;
        A(1,2) = z;
        A(1,3) = z;
        A(2,0) = 1;
        A(2,1) = 1;
        A(3,1) = 1;
        A(3,4) = 2*x1-6;
        A(3,5) = 2*x2-2;
        A(4,3) = 2*x1-6;
        A(4,4) = 2-2*lambda+2*v;
        A(4,6) = 2-2*x1;
        A(5,3) = 2*x2-2;
        A(5,5) = 2-2*lambda+2*v;
        A(5,6) = 2-2*x2;
        A(6,4) = 2-2*x1;
        A(6,5) = 2-2*x2;

        double hx = x1*x1 + x2*x2 - 6*x1  - 2*x2 + 10;
        double gx = x1*x1 + x2*x2 - 2*x1 - 2*x2 - 2;

        b[0] = mi - s*p;
        b[1] = mi - z*(v+p);
        b[2] = -s - z + hmax - hmin;
        b[3] = - hx  - z + hmax;
        b[4] = -(2*x1 - 4) + lambda *(2*x1 - 2) - v*(2*x1 - 6);
        b[5] = -(2*x2 - 8) + lambda *(2*x2 - 2) - v*(2*x2 - 2);
        b[6] = gx;
};

//Objective function

double fx(double x1, double x2)
{
        return  x1*x1+x2*x2-4*x1-8*x2+20;
};

//Constraints

double v1()
{
        double hx = x1*x1 + x2*x2 - 6*x1  - 2*x2 + 10;
        double gx = x1*x1 + x2*x2 - 2*x1  - 2*x2 - 2;
        return __max( __max(hmin - hx, hx-hmax), fabs(gx));
};

//Convergence test

double v2()
{
        double norm2x = sqrt(x1*x1 + x2*x2);
        double norm2lambda = fabs(lambda);
        double norm2v = fabs(v);
        df[0] = 2*x1-4;
        df[1] = 2*x2 -8;
        Jg[0] = 2*x1-2;
        Jg[1] = 2*x2-2;
        Jh[0] = 2*x1-6;
        Jh[1] = 2*x2-2;

        return __max(fabs(df[0] - Jg[0]*lambda + Jh[0]*v), fabs(df[1] - Jg[1]*lambda
+ Jh[1]*v) )
                /(1+norm2x+norm2lambda+norm2v);
};

double v3()
{
        double norm2x = sqrt(x1*x1 + x2*x2);
        return ro/(1+norm2x);
};

double v4()
{
        return fabs(fx(x1,x2) - fx(x1p, x2p))/(1+fabs(fx(x1,x2)));
};
```