This is a note of paper "A Newton-Type Method for Positive-Semidefinite Linear Complementarity Problems", which provides a newton method of solving LCP problem which we meet in Ben. Moll (2017). The file “LCP.m” utilizes a slightly modified newton method, which could accelerate the algorithm, while the main idea is not changed.

LCP problem:

\[ Mx + q = y, y^T x = 0, x, y \geq 0. M \text{ is a } n \times n \text{ matrix, } x, y \text{ is } n \times 1 \text{ vector} \]

(1)

Notation: \(| \cdot |\), norm;

Define a mapping:

\[ T(w) := [(Mx + q - y)_{n \times 1}, \phi(w_1), \phi(w_2), \ldots, \phi(w_n)]^T \]

Where: \( w_i := (x_i, y_i) \) and the function: \( \phi(a, b) = \sqrt{a^2 + b^2} - a - b \). We have to note that \( \phi \) equals to zero if and only if \( ab = 0, a, b \geq 0 \).

General idea: \( w = (w_1; w_2; \ldots) \) is the solution of LCP if and only if \( |T(w)| = 0 \rightarrow T(w)_i = 0 \). Since, \( Mx + q = y \ (n \times 1) \), and \( y^T x = 0 \iff \phi(a, b) = 0 \).

As some elements will be zero and \( \nabla T(x, y) \) may be singular/non-exist. Therefore, we introduce \( G(w) \):

\[ G(w) = \begin{bmatrix} M & \text{diag}(-1) \\ \text{diag}(\mu(w_i)) & \text{diag}(\nu(w_i)) \end{bmatrix}, \]

(2)

where \( \mu(w_i) = -1 + \frac{x_i}{\sqrt{x_i^2 + y_i^2}}, \nu(w_i) = -1 + \frac{y_i}{\sqrt{x_i^2 + y_i^2}} \)

(3)

Algorithm:

Step 0. Choose \( w^0 \) and \( \lambda \in (0,1) \). Set \( k = 0 \).

Step 1. If \( |T(w)| = 0 \), stop.

Step 2. Determine \( \tilde{w}^k \) as solution of:

\[ G(\tilde{w}^k)(w - \tilde{w}^k) + T(\tilde{w}^k) = 0 \]

Step 3. Update \( w \).
**Algorithm A.**

Step 0. Choose $w^0 \in R^{2n}$ and $\lambda \in (0, 1)$. Set $k := 0$.

Step 1. If $\| T(\tilde{w}^k) \| = 0$, then stop.

Step 2. Determine $\hat{w}^k \in R^{2n}$ as solution of

$$G(\tilde{w}^k)(w - \hat{w}^k) + T(\tilde{w}^k) = 0.$$ 

Step 3. Find the largest number $t_k \in \{ (1/2)^j | j \in N \}$ such that

$$\| T(\tilde{w}^k + t_k(\hat{w}^k - \tilde{w}^k)) \| \leq (1 - \lambda t_k) \| T(\tilde{w}^k) \|.$$ 

Step 4. Set $w^{k+1} := \tilde{w}^k + t_k(\hat{w}^k - \tilde{w}^k)$ and $k := k + 1$. Go to Step 1.

Reference:


function x = LCP(M,q,l,u,x0,display)
%LCP Solve the Linear Complementarity Problem.
%
% USAGE
%   x = LCP(M,q) solves the LCP
%   x >= 0
%   Mx + q >= 0
%   x'(Mx + q) = 0
%
%   x = LCP(M,q,l,u) solves the generalized LCP (a.k.a MCP)
%   1 < x < u        =>  Mx + q = 0
%   x = u            =>  Mx + q < 0
%   l = x            =>  Mx + q > 0
%
%   x = LCP(M,q,l,u,x0,display) allows the optional initial value 'x0' and
%   a binary flag 'display' which controls the display of iteration data.
%
% Parameters:
%   tol       - Termination criterion. Exit when 0.5*phi(x)'*phi(x) < tol.
%   mu        - Initial value of Levenberg-Marquardt mu coefficient.
%   mu_step   - Coefficient by which mu is multiplied / divided.
%   mu_min    - Value below which mu is set to zero (pure Gauss-Newton).
%   max_iter  - Maximum number of (successful) Levenberg-Marquardt steps.
%   b_tol     - Tolerance of degenerate complementarity: Dimensions where
%                max( min(\abs(x-l),\abs(u-x)), abs(phi(x)) ) < b_tol
%                are clamped to the nearest constraint and removed from
%                the linear system.
%
% ALGORITHM
%   This function implements the semismooth algorithm as described in [1],
%   with a least-squares minimization of the Fischer-Burmeister function using
%   a Levenberg-Marquardt trust-region scheme with mu-control as in [2].
%
% [1] A. Fischer, A Newton-Type Method for Positive-Semidefinite Linear
%     Complementarity Problems, Journal of Optimization Theory and
%
%
% Copyright (c) 2008, Yuval Tassa
% tassa at alice dot huji dot ac dot il
tol = 1.0e-12;
mu = 1e-3;
mu_step = 5;
mu_min = 1e-5;
max_iter = 20;
b_tol = 1e-6;

n = size(M,1);

if nargin < 3 || isempty(l)
l = zeros(n,1);
if nargin < 4 || isempty(u)
u = inf(n,1);
if nargin < 5 || isempty(x0)
x0 = min(max(ones(n,1),l),u);
if nargin < 6
display = false;
end
end
end

lu = [l u];
x = x0;

[psi,phi,J] = FB(x,q,M,l,u);
new_x = true;
warning off MATLAB:nearlySingularMatrix
for iter = 1:max_iter

if new_x
    [mlu,ilu] = min([abs(x-l),abs(u-x)],[],2);
    bad = max(abs(phi),mlu) < b_tol;
    psi = psi - 0.5*phi(bad)'*phi(bad);
    J = J(~bad,~bad);
    phi = phi(~bad);
    new_x = false;
    nx = x;
    nx(bad) = lu(find(bad)+(ilu(bad)-1)*n);
end

H = J'*J + mu*speye(sum(~bad));
Jphi = J'*phi;

d = -H\Jphi;

Initialize the output
Clear the singular part and consider the optimization problem in a subspace
Intuitively, we should follow [1]'s newton method, so we need to find out the Jacobian. [2] develops a new method with drift mu, so H is kind of new Jacobian.
Newton Update

\[
\text{nx(~bad)} = \text{x(~bad)} + d;
\]
\[
[\text{npsi}, \text{nphi}, \text{nJ}] = \text{FB(nx,q,M,l,u)};
\]
\[
r = (\text{psi} - \text{npsi}) / -(\text{Jphi}'*d + 0.5*d'*H*d); \quad \% \text{actual reduction} / \text{expected reduction}
\]

if \( r < 0.3 \)  \% small reduction, increase mu
  \[
  \text{mu} = \max(\mu*\mu_{\text{step}}, \mu_{\text{min}});
  \]
eend
if \( r > 0 \)  \% some reduction, accept nx
  \[
  \text{x} = \text{nx}; \quad \psi = \text{npsi}; \quad \phi = \text{nphi}; \quad \text{J} = \text{nJ}; \quad \text{new_x} = \text{true};
  \]
  if \( r > 0.8 \)  \% large reduction, decrease mu
    \[
    \text{mu} = \mu/\mu_{\text{step}} \times (\mu > \mu_{\text{min}});
    \]
eend
eend
if display
  \[
  \text{disp(sprintf('iter = %2d, psi = %3.0e, r = %3.1f, mu = %3.0e', iter, psi, r, mu));}
  \]
eend
if psi < tol
  break;
eend
eend
warning on MATLAB:nearlySingularMatrix
\[
\text{x} = \min(\max(\text{x}, \text{l}), \text{u});
\]

function \([\psi, \phi, J]\) = \text{FB}(x, q, M, l, u)
\[
\text{n} = \text{length}(x);
\]
\[
\text{Zl} = \text{l} > -\infty \& \text{u} = \infty;
\]
\[
\text{Zu} = \text{l} = -\infty \& \text{u} < \infty;
\]
\[
\text{Zlu} = \text{l} > -\infty \& \text{u} < \infty;
\]
\[
\text{Zf} = \text{l} = -\infty \& \text{u} = \infty;
\]
\[
\text{a} = \text{x}; \quad \text{b} = \text{M*x+q};
\]
\[
\text{a(Zl)} = \text{x(Zl)} - \text{l(Zl)};
\]
\[
\text{a(Zu)} = \text{u(Zu)} - \text{x(Zu)};
\]
\[
\text{b(Zu)} = -\text{b(Zu)};
\]

In addition, I guess it is common to use adjustable mu to accelerate the algorithm and this criteria can be found in some convex optimization book.
if any(Zlu)
    nt     = sum(Zlu);
    at     = u(Zlu)-x(Zlu);
    bt     = -b(Zlu);
    st     = sqrt(at.^2 + bt.^2);
    a(Zlu) = x(Zlu)-l(Zlu);
    b(Zlu) = st -at -bt;
end

s        = sqrt(a.^2 + b.^2);
phi      = s - a - b;
phi(Zu)  = -phi(Zu);
phi(Zf)  = -b(Zf);

psi      = 0.5*phi'*phi;

if nargout == 3
    if any(Zlu)
        M(Zlu,:) = - sparse(1:nt,find(Zlu),at./st-ones(nt,1),nt,n) -
                   sparse(1:nt,1:nt,bt./st-ones(nt,1))*M(Zlu,:);
    end
    da       = a./s-ones(n,1);
    db       = b./s-ones(n,1);
    da(Zf)   = 0;
    db(Zf)   = -1;
    J        = sparse(1:n,1:n,da) + sparse(1:n,1:n,db)*M;
end