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:

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

Notation: $|\cdot|$, norm;

Define a mapping:

$$\mathrm{T}(\mathrm{w}) \coloneqq\left[(\mathrm{Mx}+\mathrm{q}-\mathrm{y})_{\mathrm{n} \times 1}, \phi\left(\mathrm{w}_1\right), \phi\left(\mathrm{w}_2\right), \ldots, \phi\left(\mathrm{w}_{\mathrm{n}}\right)\right]^T$$

Where: $\mathrm{w}_{\mathrm{i}} \coloneqq\left(x_i, y_i\right)$ and the function: $\phi(\mathrm{a}, \mathrm{b}) \coloneqq \sqrt{a^2+b^2}-a-b$. We have to note that ϕ equals to zero if and only if $\mathrm{ab}=0, \mathrm{a}, \mathrm{b} \geq 0$.

General idea: $\mathrm{w}=\left(\mathrm{w}_1 ; \mathrm{w}_2 ; \ldots\right)$ is the solution if and only if $|\mathrm{T}(\mathrm{w})|=0$.

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

$$\mathrm{G}(\mathrm{w})=\begin{bmatrix} M & diag(-1) \\ diag\big(\mu(w_i)\big) & diag\big(\nu(w_i)\big) \end{bmatrix},$$

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

Algorithm:

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

Step 1. If $|\mathrm{T}(\mathrm{w})|=0$, stop.

Step 2. Determine $\widehat{\mathrm{w}}^k$ as solution of:

$$\mathrm{G}(\overline{\mathrm{w}}^{\mathrm{k}})(\mathrm{w}-\overline{\mathrm{w}}^{\mathrm{k}})+T(\overline{\mathrm{w}}^{\mathrm{k}})=0$$

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 $\|T(\bar{w}^k)\| = 0$, then stop.

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

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

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

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

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

```
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)
%
%   l < 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 (succesful) 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
%   Applications: Vol. 86, No. 3, pp. 585-608, 1995.
%
%   [2] M. S. Bazarraa, H. D. Sherali, and C. M. Shetty, Nonlinear
%   Programming: Theory and Algorithms. John Wiley and Sons, 1993.
%
%   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
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 Method

```
   nx(~bad)       = x(~bad) + d;
   [npsi,nphi,nJ] = FB(nx,q,M,l,u);
   r              = (psi - npsi)  / -(Jphi'*d + 0.5*d'*H*d);  % actual reduction
/ expected reduction
   if r < 0.3          % small reduction, increase mu
      mu = max(mu*mu_step,mu_min);
   end
   if r > 0            % some reduction, accept nx
      x     = nx;
      psi   = npsi;
      phi   = nphi;
      J     = nJ;
      new_x = true;
      if r > 0.8       % large reduction, decrease mu
         mu = mu/mu_step * (mu > mu_min);
      end
   end
   if display
      disp(sprintf('iter    =    %2d,    psi    =    %3.0e,    r    =    %3.1f,    mu
= %3.0e',iter,psi,r,mu));
   end
   if psi < tol
      break;
   end
end
warning on MATLAB:nearlySingularMatrix
x = min(max(x,l),u);
```

Newton Update

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.

```
function [psi,phi,J] = FB(x,q,M,l,u)
n     = length(x);
Zl    = l >-inf & u==inf;
Zu    = l==-inf & u <inf;
Zlu   = l >-inf & u <inf;
Zf    = l==-inf & u==inf;

a     = x;
b     = M*x+q;

a(Zl) = x(Zl)-l(Zl);

a(Zu) = u(Zu)-x(Zu);
b(Zu) = -b(Zu);
```

Four possible boundaries

More details from note.

```
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;
```

G-Matrix depends on J

```
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
```