LMMCP — A LEVENBERG-MARQUARDT-TYPE MATLAB[1] SOLVER FOR MIXED COMPLEMENTARITY PROBLEMS

Christian Kanzow and Stefania Petra

University of Würzburg
Institute of Applied Mathematics and Statistics
Am Hubland
97074 Würzburg
Germany

e-mail: kanzow@mathematik.uni-wuerzburg.de
petra@mathematik.uni-wuerzburg.de

October 10, 2005

Abstract: We describe the implementation of a nonsmooth Levenberg-Marquardt-type method for mixed complementarity problems under the MATLAB environment. The resulting software is called LMMCP – Levenberg-Marquardt Mixed Complementarity Problem solver. We give the basic ideas of this solver and discuss the importance of certain parameters. Numerical results for the MCPLIB test problem collection are also given.

Keywords: Mixed complementarity problems, Levenberg-Marquardt method, MATLAB program.

[1]MATLAB is a registered trademark of The MathWorks, Inc.

1 Introduction

The mixed complementarity problem, MCP for short, is one of the fundamental problems in optimization. The theoretical background and many numerical methods for the solution of this problem are described in the recent books [8, 9] by Facchinei and Pang. Other books dealing, in particular, with the special case of a complementarity problem, include [4, 16, 19]. Several important applications of MCPs may be found in the survey article [12] by Ferris and Pang.

The most convenient way to formulate an MCP is via a variational inequality: Given a continuously differentiable function $F : \mathbb{R}^n \to \mathbb{R}^n$ and a nonempty, closed and convex set $X \subseteq \mathbb{R}^n$, the variational inequality problem consists in finding a point $x^* \in X$ such that

$$F(x^*)^T(x - x^*) \geq 0 \quad \forall x \in X.$$

If the feasible set X is a box of the form $X = [l, u]$ with lower bounds $l = (l_1, \dots, l_n)^T$ and upper bounds $u = (u_1, \dots, u_n)^T$ satisfying $-\infty \leq l_i < u_i \leq +\infty$ for all $i \in \{1, \dots, n\}$, we obtain the mixed complementarity problem. Writing down the corresponding KKT optimality conditions (see [8] for further details), it is easy to see that $x^* \in [l, u]$ is a solution of MCP if and only if exactly one of the following conditions hold:

$$\begin{aligned} x_i^* = l_i \quad &\text{and} \quad F_i(x^*) > 0, \\ x_i^* = u_i \quad &\text{and} \quad F_i(x^*) < 0, \\ x_i^* \in [l_i, u_i] \quad &\text{and} \quad F_i(x^*) = 0. \end{aligned}$$

A further reformulation of these conditions will be given in Section 2 where we review the basic ideas of our approach.

Commercial software for the solution of MCPs is available: There is a solver called MILES by Rutherford [20] and the very successful PATH code by Dirkse and Ferris [6], see also [11, 10] for some later versions. Both codes are based on the Josephy-Newton idea, combined with many enhancements in order to improve the overall performance. A comparison of these codes with a number of other algorithms may be found in the paper [2] by Billups, Dirkse, and Ferris.

In this paper, we present the main ideas of a relatively simple MATLAB program called LMMCP for the solution of MCPs. This program is the outcome of the authors' two recent papers [17, 18] and combines some ideas from these two related works. Basically, the method uses a nonsmooth reformulation of the mixed complementarity problem as an overdetermined system of equations (or a nonlinear least squares problem with zero residual) and applies a (projected) Levenberg-Marquardt-type approach to this reformulation of the original MCP. The method has a strong theoretical background which, however, is not the main focus of this manuscript.

Here we want to describe the basic idea of our MATLAB solver LMMCP and we give some details regarding the choice of certain parameters that the reader might want to change in order to optimize the code for his particular example. The main ideas and ingredients of the LMMCP program are therefore given in Section 2, whereas the details regarding some of its parameters are presented in Section 3. A summary of the numerical results obtained by LMMCP being applied to the MCPLIB test problem collection are given in Section 4, and we close with some final remarks in Section 5.

2 Basic Idea of LMMCP

We first recall a reformulation of the MCP that was used in [17, 18] and then present the main conceptual ingredients of the LMMCP code.

In order give the reformulation of MCPs, let us introduce the following partition of the index set $I := \{1, \dots, n\}$:

$$\begin{aligned} I_l &:= \{i \in I \mid -\infty < l_i < u_i = \infty\}, \\ I_u &:= \{i \in I \mid -\infty = l_i < u_i < \infty\}, \\ I_{lu} &:= \{i \in I \mid -\infty < l_i < u_i < \infty\}, \\ I_f &:= \{i \in I \mid -\infty = l_i < u_i = \infty\}. \end{aligned}$$

Hence the indices l, u, and lu indice which bounds are finite, where I_f is the set of free variables x_i where both the lower bounds l_i and the upper bounds u_i are infinite.

Furthermore, let $\phi : \mathbb{R}^2 \to \mathbb{R}$ denote the Fischer-Burmeister function

$$\phi(a, b) := \sqrt{a^2 + b^2} - a - b,$$

which has the interesting property that

$$\phi(a, b) = 0 \Longleftrightarrow a \geq 0, b \geq 0, ab = 0, \tag{1}$$

see [13]. We now define the operator $\Phi : \mathbb{R}^n \to \mathbb{R}^{2n}$ componentwise as follows ($i = 1, \dots, n$):

$$\Phi_i(x) := \begin{cases} \lambda\phi(x_i - l_i, F_i(x)) & \text{if } i \in I_l, \\ -\lambda\phi(u_i - x_i, -F_i(x)) & \text{if } i \in I_u, \\ \lambda\phi(x_i - l_i, \phi(u_i - x_i, -F_i(x))) & \text{if } i \in I_{lu}, \\ -\lambda F_i(x) & \text{if } i \in I_f, \end{cases}$$

$$\Phi_{n+i}(x) := \begin{cases} (1-\lambda)\phi_+(x_i - l_i, F_i(x)) & \text{if } i \in I_l, \\ (1-\lambda)\phi_+(u_i - x_i, -F_i(x)) & \text{if } i \in I_u, \\ (1-\lambda)(\phi_+(x_i - l_i, F_i(x)) + \phi_+(u_i - x_i, -F_i(x))) & \text{if } i \in I_{lu}, \\ -(1-\lambda)F_i(x) & \text{if } i \in I_f. \end{cases}$$

Then it was noted in [17] that the overdetermined system of equations

$$\Phi(x) = 0 \tag{2}$$

is equivalent to the MCP. Obviously, the same holds for the box constrained reformulation

$$\Phi(x) = 0, x \in [l, u] \tag{3}$$

which is the basis of the method from [18]. Note that we can use any function ϕ with the property (1) in order to get an unconstrained or box constrained reformulation of the MCP. In the moment, however, we use the Fischer-Burmeister function for ϕ in our implementation. An interesting consequence of using the Fischer-Burmeister function is the fact that the corresponding merit function

$$\Psi(x) := \frac{1}{2}\|\Phi(x)\|^2 \tag{4}$$

is continuously differentiable although Φ itself is nonsmooth.

Basically, the LMMCP code consists of two phases: Phase I is a preprocessor, and Phase II contains the main algorithm. We give some more details in the following:

Phase I: In this preprocessing phase, we use some iterations of the local projected Levenberg-Marquardt method from [18] which is based on the constrained reformulation (3). By default, at most 20 iterations are allowed in this preprocessing phase. There is no globalization used in this phase. Nevertheless, according to our experience, this method has a very good behaviour, and most problems will actually be solved in this phase, especially those examples that are viewed as being simple problems. If we are not able to solve a problem in this phase, we switch to Phase II.

Phase II: This is the main program using the Levenberg-Marquardt method from [17] with a line search globalization based on the unconstrained reformulation (2). This phase is started with the original starting point x^0. We use a nonmonotone line search (see [14]) together with a watchdog stabilization technique (see [15]), i.e., if the best function value found so far has not been reduced sufficiently within a fixed number of iterations, we restart from that point using a monotone line search.

The computation of our search direction requires the evaluation of a generalized Jacobian (see [3]) of the nonsmooth operator Φ. To this end, we use the strategy described in [1] which, in turn, in based on the one given in [5] for the standard nonlinear complementarity problems. If the function ϕ is replaced by another function having the property (1), however, the evaluation of this generalized Jacobian has to be modified, too.

3 M-Files and Important Parameters

Here we shortly describe the structure of the program and give the default values of some parameters and discuss the role of these parameters.

The MATLAB solver LMMCP can be downloaded from the first author's homepage using the URL

`http://www.mathematik.uni-wuerzburg.de/~kanzow/`

and clicking on the `software` button. Then there is a file called

`LMMCP.zip`.

Downloading and unzipping this file will result in the following list of M-files:

```
LMMCP_V10.m
LMMCP_MCPLIB_V10.m
Phi3MCPPFB.m
DPhi3MCPPFB.m
josephy.m
Djosephy.m
Sjosephy.m
```

The main program is contained in `LMMCP_V10.m`. The related code `LMMCP_MCPLIB_V10.m` is almost a copy of this program except that it contains some special lines for solving test examples from the MCPLIB test problem collection, see [7]. Depending on the local installation of this collection, however, one has to change a few lines containing the path for this particular collection of test examples. These two main programs call the two M-files `Phi3MCPPFB.m` and `DPhi3MCPPFB.m`, where the former is used to evaluate the function Φ at the current iterate, whereas the latter calculates a generalized Jacobian of Φ at the current point. The M-files `josephy`, `Djosephy`, and `Sjosephy` contain the data of a simple example and provide the function value $F(x)$, the Jacobian $F'(x)$, and a starting point x^0 together with the lower and upper bounds l_i, u_i, respectively. Any other example that should be solved using `LMMCP_V10.m` needs similar M-files, so that the user has to provide, for each test problem, three MATLAB-files.

The program can be tested by typing `LMMCP_V10.m` in the MATLAB environment. Then the program will ask for the name of an example. Typing `josephy`, for example, will result in the following output:

```
   k               Psi(x)               || DPsi(x) ||    stepsize
===================================================================
******************** Output at starting point ********************
   0          0.022810535808             2.1113411968
************************* Preprocessor ***************************
   1          0.000017271137             0.0481257223   1.0000000
   2          0.000000000030             0.0000630524   1.0000000
```

This output shows that the program was able to solve this problem in just 2 iterations (both in Phase I). The four columns have the the following (obvious) meanings: The first column gives the current iteration counter, the second column the function value $\Psi(x^k)$ at the current iterate, the third column provides the norm $\|\nabla\Psi(x^k)\|$ at the point x^k, and the last column give the stepsize t_k. Note that this stepsize is always equal to one in Phase I, so this column becomes important only in Phase II where this stepsize is typically different from one.

Typing 'x' in the MATLAB environment gives the approximate solution of the problem that was solved before. Especially for the previous `josephy` example, we get the following vector:

```
>> x

x =

    1.2247
         0
    0.0000
    0.5000
```

We next give some details regarding some of the more important parameters in the LMMCP code.

Choice of λ: The mapping Φ and, therefore, both reformulations (2) and (3) of the MCP depends on a parameter λ that is called `lambda1` is our code. Theoretically, this parameter can be any number from the interval $(0, 1]$ (note that zero is excluded here). Numerically, we found 0.1 to be a good choice, and therefore we use the value

$$\texttt{lambda1} = \texttt{0.1}$$

by default. However, the choice of this parameter usually has a great influence on the numerical behaviour of the overall algorithm, and a different value of this parameter might give significantly better results for particular problems. We suggest to change this parameter first whenever our LMMCP code is not able to solve a problem.

Switch from Phase I to Phase II: Basically, there are two parameters that control the switch from Phase I to Phase II of our algorithm. They are called `preprocess` and `presteps`. By default, we use

$$\texttt{preprocess} = \texttt{1} \quad \text{and} \quad \texttt{presteps} = \texttt{20}.$$

Setting `preprocess=1` simply says that we want to apply our preprocessor from Phase I first. If we assign any other number to this parameter, then our LMMCP solver immediately goes to the main program in Phase II. Moreover, `presteps=20` means that we allow at most 20 iterations in Phase I. Thereafter we switch to Phase II. Changing the parameter `presteps` sometimes gives completely different results.

Termination parameters: There are mainly three parameters that are used to terminate the iteration, namely `eps2`, `kmax`, and `tmin`, whose default values are

$$\texttt{eps2} = 10^{-10}, \quad \texttt{kmax} = 500, \quad \texttt{tmin} = 10^{-12}.$$

If the current iterate x^k gives a function value $\Psi(x^k)$ of less than `eps2`, then we stop our iteration successfully with an approximate solution. This parameter might be changed if a higher or lower accuracy is required. The parameter `kmax` is used as the maximum number of iterations that is allowed to be taken by our LMMCP algorithm (note that we count the iterations from Phase I and Phase II together). In general, it seems that our method is not able to solve a problem in more than 500 iterations if it was not able to solve it within the first 500 steps, so we do not suggest to change this parameter. Finally, the parameter `tmin` is a safeguard for the stepsize t_k used in Phase II: If t_k becomes less than `tmin`, we terminate with an error message. Changing the default value of `tmin` slightly usually does not change much regarding the overall performance of the method.

Choice of some stepsize parameters: There are a several parameters that play an important role for our choice of a suitable stepsize t_k in Phase II. Here we only mention the parameters `m`, `kwatch`, and `watchdog`, whose default values are

$$\texttt{m} = 10, \quad \texttt{kwatch} = 20, \quad \texttt{watchdog} = 1.$$

The parameter `m` tells the nonmonotone line search (Armijo) rule to use the last ten function values in order to decide whether or not to accept the new iterate. The nonmonotone line search is combined with a watchdog strategy. This means that we go back to the best point found so far if we are not able to get a significant reduction of the function value within `kwatch` steps. Note that the value of `kwatch` should not be smaller than the value of `m`. Furthermore, if the watchdog strategy should not be used, simply set the parameter `watchdog` to any number different from one.

There are some other parameters used in our LMMCP code, in particular, there are some parameters regarding the choice of a Levenberg-Marquardt parameter and some parameters for an update of an implicitly used trust-region radius in Phase I. However, we do not give the details here, the interested reader may have a look into the program.

4 Numerical Results

In this section, we shortly summarize the numerical results that we obtained using our LMMCP code being applied to the MCPLIB test problem collection, see [7]. We report our results in two tables: Table 1 contains the results for the smaller problems, and

Table 2 gives the corresponding results for the larger problems. For each test example, we give the name of the problem, its dimension n, the function value $\Psi(x^0)$ at the initial iterate x^0, the number Nit of iterations the method used until termination, the function value $\Psi(x^f)$ at the final iterate x^f, and the norm $\|\nabla\Psi(x^f)\|$ of the gradient of Ψ at x^f.

Table 1: Numerical results for MCPLIB test problems

Problem	Dim	$\Psi(x^0)$	Nit	$\Psi(x^f)$	$\|\nabla\Psi(x^f)\|$
badfree	5	4.6000e–01	4	1.5896e–14	2.5216e–08
bertsekas	15	3.9360e–03	14	3.9695e–15	3.6057e–07
billups	1	3.4512e–05	50	2.2044e–12	7.6273e–06
choi	13	7.7090e–03	5	2.6496e–16	9.3824e–10
colvdual	20	5.4880e+01	17	1.6696e–11	9.0001e–05
colvnlp	15	6.2076e+01	6	4.8859e–16	5.1459e–08
cycle	1	5.1738e+01	4	8.9220e–12	4.2242e–07
degen	2	1.0000e–01	4	3.1519e–17	7.9396e–10
duopoly	63	2.1325e+02	—	—	—
ehl_k40	41	1.0422e+04	12	1.5114e–13	1.8418e–04
ehl_k60	61	3.7975e+04	15	3.1039e–11	3.9762e–04
ehl_k80	81	9.3630e+04	14	9.4744e–13	3.8724e–03
ehl_kost	101	1.8790e+05	17	5.6519e–12	1.5125e–02
electric	158	2.6097e+08	48	4.3937e–11	7.5483e–02
explcp	16	3.2000e–01	4	7.4076e–14	3.8491e–08
forcebsm	184	3.9442e+03	259	3.0957e–12	2.4894e–07
forcedsa	186	3.9487e+03	45	2.9715e–16	2.4378e–09
freebert	15	1.5098e+04	13	1.9019e–13	2.7216e–06
gafni	5	1.3004e+03	10	6.4207e–13	2.8455e–05
games	16	6.0066e+01	13	1.2298e–12	1.6448e–05
hanskoop	14	1.1860e+02	14	5.6144e–13	1.6009e–07
hydroc06	29	1.7666e+05	7	7.7161e–18	1.3727e–05
hydroc20	99	4.1044e+05	10	2.6114e–15	1.2675e–04
jel	6	9.5612e+02	8	5.3611e–12	2.9118e–05
josephy	4	2.2811e–02	2	2.9891e–11	6.3052e–05
kojshin	4	2.2811e–02	2	3.0042e–11	6.3208e–05
mathinum	3	6.2154e+02	5	4.5040e–17	2.1547e–08
mathisum	4	5.2165e+00	8	2.1995e–16	1.5101e–08
methan08	31	6.4638e+06	5	6.3205e–19	2.4385e–07
nash	10	5.4263e+02	4	2.3546e–19	7.3984e–09
ne-hard	3	1.1559e+04	19	1.8429e–11	7.7746e–05
pgvon106	106	1.5367e+02	41	6.1101e–12	8.9650e–07
pies	42	5.2678e+08	46	3.3569e–11	1.4099e-03
powell	16	6.8071e–04	4	5.6591e–11	4.0948e–06

Table 1: Numerical results for MCPLIB test problems (continued)

Problem	Dim	$\Psi(x^0)$	Nit	$\Psi(x^f)$	$\|\nabla\Psi(x^f)\|$
powell_mcp	8	9.3167e+01	2	2.7283e–13	5.4316e–06
qp	4	3.3000e+00	2	1.6034e–31	7.5011e–16
scarfanum	13	6.9949e–05	3	3.6051e–12	2.4848e–06
scarfasum	14	6.9949e–05	3	3.6049e–12	1.9089e–06
scarfbsum	40	1.1232e+02	46	9.2880e–11	2.8655e–03
shubik	45	1.6389e–01	266	9.5806e–13	8.2964e–03
simple-ex	17	9.5616e+00	73	7.3173e–13	9.4441e–07
simple-red	13	2.2508e+02	10	2.1736e–11	3.1994e–06
sppe	27	1.2169e+02	3	4.2516e–11	1.2021e–04
tinloi	146	4.0018e–01	6	7.6395e–12	4.2206e–04
tobin	42	3.2365e+00	2	1.4746e–14	9.5573e–06

Table 2: Numerical results for MCPLIB test problems

Problem	Dim	$\Psi(x^0)$	Nit	$\Psi(x^f)$	$\|\nabla\Psi(x^f)\|$
bert_oc	5000	5.1294e+01	5	8.4889e–14	4.2054e–10
bratu	5625	7.8870e+00	12	1.7099e–14	1.5884e–08
bishop	1645	2.1577e+11	—	—	—
lincont	419	3.9560e+03	41	5.3360e–18	9.6787e–07
obstacle	2500	3.3714e–04	6	8.6923e–11	1.6423e–06
opt_cont	288	8.6940e+02	5	3.1396e–18	1.1722e–08
opt_cont31	1024	2.4359e+02	5	6.0898e–16	1.7810e–07
opt_cont127	4096	7.8674e+01	5	2.2750e–18	2.9798e–10
opt_cont255	8192	5.1845e+01	5	8.1394e–12	4.8056e–07
opt_cont511	16384	3.8488e+01	6	1.7258e–17	1.5287e–08
trafelas	2904	5.1250e+03	163	6.5673e–17	1.5852e–09

The previous two tables indicate that LMMCP is able to solve almost all problems including a number of very difficult ones like `electric`, `forcebsm`, `forcedsa`, `ne-hard`, `pgvon106`, `simple-ex`, and `lincont`. We have failures on two problems only, namely `duopoly` and `bishop` which, however, can be solved with other parameter settings.

5 Conclusions

We gave some details of our implementation of LMMCP, a MATLAB program for the solution of mixed complementarity programs based on a nonsmooth reformulation as an overdetermined system of equations. We believe that this is a very robust solver, however, the current implementation is version 1.0, and there are a lot of possibilities for improvements. For example, we may replace the Fischer-Burmeister function ϕ by another function with similar properties, or we may use automatic restarts when it seems that our program is not able to solve a certain test examples. These modifications will certainly be incorporated in future versions of this program.

References

[1] S.C. Billups: *Algorithms for complementarity problems and generalized equations.* Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, Madison, WI, August 1995.

[2] S.C. Billups, S.P. Dirkse, and M.C. Ferris: *A comparison of large scale mixed complementarity problem solvers.* Computational Optimization and Applications 7, 1997, pp. 3–25.

[3] F.H. Clarke: *Optimization and Nonsmooth Analysis.* John Wiley & Sons, New York, NY, 1983 (reprinted by SIAM, Philadelphia, PA, 1990).

[4] R.W. Cottle, J.-S. Pang, and R.E. Stone: *The Linear Complementarity Problem.* Academic Press, Boston, 1992.

[5] T. De Luca, F. Facchinei and C. Kanzow: *A semismooth equation approach to the solution of nonlinear complementarity problems.* Mathematical Programming 75, 1996, pp. 407–439.

[6] S.P. Dirkse and M.C. Ferris: *The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems.* Optimization Methods and Software 5, 1995, pp. 123–156.

[7] S.P. Dirkse and M.C. Ferris: *MCPLIB: A collection of nonlinear mixed complementarity problems.* Optimization Methods and Software 5, 1995, pp. 319–345.

[8] F. Facchinei and J.-S. Pang: *Finite-Dimensional Variational Inequalities and Complementarity Problems, Volume I.* Springer, New York, 2003.

[9] F. Facchinei and J.-S. Pang: *Finite-Dimensional Variational Inequalities and Complementarity Problems, Volume II.* Springer, New York, 2003.

[10] M.C. Ferris, C. Kanzow, and T.S. Munson: *Feasible descent algoritms for mixed complementarity problems.* Mathematical Programming 86, 1999, pp. 475–497.

[11] M.C. Ferris and T.S. Munson: *Interfaces to PATH 3.0: Design, implementation and usage.* Computational Optimization and Applications 12, 1999, pp. 207–227.

[12] M.C. Ferris and J.S. Pang: *Engineering and economic applications of complementarity problems.* SIAM Review, 39, 1997, pp. 669–713.

[13] A. Fischer: *A special Newton-type optimization method.* Optimization 24, 1992, pp. 269–284.

[14] L. Grippo, F. Lampariello, and S. Lucidi: *A nonmonotone line search technique for Newton's method.* SIAM Journal on Numerical Analysis 23, 1986, pp. 707–716.

[15] L. Grippo, F. Lampariello, and S. Lucidi: *A class of nonmonotone stabilization methods in unconstrained optimization.* Numerische Mathematik 59, 1991, pp. 779–805.

[16] G. Isac: *Complementarity Problems.* Lecture Notes in Mathematics 1528, Springer, Berlin, 1992.

[17] C. Kanzow and S. Petra: *On a semismooth least squares formulation of complementarity problems with gap reduction.* Optimization Methods and Software 19, 2004, pp. 507–525.

[18] C. Kanzow and S. Petra: *Projected filter trust region methods for a semismooth least squares formulation of mixed complementarity problems.* Preprint, Institute of Applied Mathematics and Statistics, University of Wuerzburg, September 2005.

[19] K.G. Murty: *Linear Complementarity, Linear and Nonlinear Programming.* Heldermann, Berlin, 1988.

[20] T.F. Rutherford: *MILES: A mixed inequality and nonlinear equation solver.* Working Paper, Department of Economics, University of Colorado, Boulder, 1993.