ACCESSING REALISTIC MIXED COMPLEMENTARITY PROBLEMS WITHIN MATLAB

Michael C. Ferris[1] and Thomas F. Rutherford[2]

[1] Computer Sciences Department
University of Wisconsin
Madison, Wisconsin 53706
The work of this author was based on research supported by
the Department of Energy grant DE-FG03-94ER61915,
the Air Force Office of Scientific Research grant F49620-94-1-0036
and the National Science Foundation grant CCR-9157632.

[2] Department of Economics
The University of Colorado
Boulder, Colorado 80309
The work of this author was based on research supported by
the Department of Energy grant DE-FG03-94ER61915.

Abstract: This paper describes a suite of programs that allow realistic complementarity problems to be accessed from within the MATLAB programming environment. The suite of programs uses data generated from the GAMS/MCP library and makes function and Jacobian evaluations available directly to the MATLAB user. All the required data is stored in a binary encoding of the problems, completely independent of GAMS. Use of the programs is described here, along with a list of all the problems that are currently available in binary form. Details on how the binary files are generated from GAMS are also given.

Key words: Mixed complementarity problems, algorithms, modeling.

1 INTRODUCTION

In recent years there has been an enormous amount of interest in developing algorithms for solving nonlinear and mixed complementarity problems. MATLAB is a package of routines for numeric computation and visualization that provides an excellent environment for prototyping such algorithms. Many basic tools, such as factorizations, matrix multiplication and addition and eigenstructure analysis are provided both in a dense and sparse matrix setting. Furthermore, other tools for matrix manipulation and visualization enable many complex algorithms to be succinctly and quickly implemented, and underlying structure to be identified.

In order to exploit these features while prototyping algorithms for complementarity problems, there needs to be a library of test problems readily available within MATLAB for testing purposes.

A suite of problems that arise from realistic applications, with reasonable size and real data, is already available within the GAMS/MCP setting as described in [3] and [7]. Other problems [11] are currently under development within this setting. This has led to broad testing of some of algorithms via the GAMS/MCP interface described in [10]. However, most of the algorithms remain largely untested in this computational setting.

This paper describes a suite of programs that are designed to make the GAMS/MCP complementarity problems readily accessible within the MATLAB [12] programming environment, without the overhead of trying to write specialized code to evaluate the underlying nonlinear functions. All of the programs are publicly available and can be obtained by anonymous ftp from `ftp.cs.wisc.edu:math-prog/matlab/` or from the authors. All the problems contained in the GAMS library and MCPLIB are freely available in the form of machine specific binary files; the data contained within these files are accessed by the aforementioned programs. The third part of the interface is a program that allows a binary file to be generated from any GAMS/MCP model. It is anticipated that this will allow the suite of problems to grow and become even more comprehensive.

The actual complementarity problems that we will generate are in the MCP format, that is, to find $z \in [l, u]$ with

$$\text{MCP}: \quad \begin{aligned} & f(z) = w - v \\ & w \geq 0,\ w^T(z - l) = 0 \\ & v \geq 0,\ v^T(u - z) = 0. \end{aligned}$$

To completely specify this problem, the function f and the values of l and u (which may possibly be infinite) must be given. For algorithm development, it is usually necessary to provide starting values for z and routines that give an evaluation of $f(z)$ and $J(z)$, the function and its Jacobian at any given point.

The interface we provide consists of a large number of problem files and several mex functions, that give the MATLAB user all of the above. Thus, the routines and the data that are provided completely specify the underlying complementarity problem. The programs essentially consist of a function and Jacobian evaluator for all of the complementarity problems. While the data necessary for these evaluations was originally obtained directly from the GAMS/MCP interface, the function evaluators are completely independent of GAMS; each problem is encoded within a single machine dependent binary file. This format was chosen essentially for speed: we believe it crucial that the function evaluations can be carried out as efficiently and quickly as possible. The binary files are freely available from the same sources as the programs.

The remainder of the paper is organized as follows. In the next section, we give a brief description of the programs, or more correctly, the `mex` files that we provide to access the problem data, and give some simple examples of how we expect these functions will be used. In the following section, a list of all the currently available problems, with some size and density information is given. Section 4 describes how the binary files were originally generated and gives details on some further advanced features of the interface. In particular, the technique a user might invoke for generating binary files from a new GAMS file is outlined. We conclude with some suggestions of possible future extensions of this work.

2 USING THE INTERFACE

The first step is to obtain the mex files and the binary problem files either from the authors or by anonymous ftp from `ftp.cs.wisc.edu:math-prog/matlab`. The interface can be used without knowing anything about the models or GAMS/MCP by just obtaining copies of the relevant binary files for your machine architecture running MATLAB. The problem files are contained in two machine dependent compressed files labeled "gamsmat" and "mcplib". The first file contains the binary files for the problems from the GAMS library, and the second contains those for MCPLIB. The original gams source files, typically having extensions `gms`, provide a valuable source of documentation for the problems. The `mex` functions can be found in the compressed file "mexfiles". The form of compression and the file extensions are different for each machine architecture. The `mex` files should be placed on the current search path of MATLAB.

Simple use of the interface consists of three `mex` files that enable MATLAB to extract from the binary file the specified starting point, the bounds on the variables, and a function and Jacobian evaluation at any given point. These mex files are `cpsetup`, `cpfun`, and `cpjac`.

These three functions are now described in more detail for a specific problem. We assume that a file `josephy.bin` is in the current MATLAB directory.

In a directory containing the relevant binary files (which should always have the extension `bin`), MATLAB should be invoked. To set up the complementarity problem in MATLAB, the following call is specified:

```
matlab> global wsvec cpfname
matlab> [z,l,u] = cpsetup('josephy');
```

This call performs the following operations. Firstly, two vectors are put into the global workspace. The vector `cpfname` contains a string which holds the name of the binary file currently in use. The vector `wsvec` is a workspace vector for use by the interface routines that is dynamically allocated within the `cpsetup` routine and is used by the other mex functions for workspace in evaluating the functions and Jacobian. Both of these vectors are generated and stored in the MATLAB workspace by `cpsetup`. After initializing these vectors, `cpsetup` determines the initial starting point `z`, and the lower and upper bounds `l,u` that specify the mixed complementarity problem (MCP). Note that the first starting point that is encountered in the GAMS problem file is the one that is returned by `cpsetup`. If any other starting point is required, the user should set it explicitly after this call, using the function `cpstart`.

After setting up the problem, to evaluate the function f of the MCP, the following call is needed:

```
matlab> f = cpfun(z);
```

The vector `f` contains the values of the function evaluated at the point `z`.

To evaluate the Jacobian `J` corresponding to the MCP, the following call is needed, which also must come after the initialization of the problem using `cpsetup`:

```
matlab> [f,J] = cpjac(z);
```

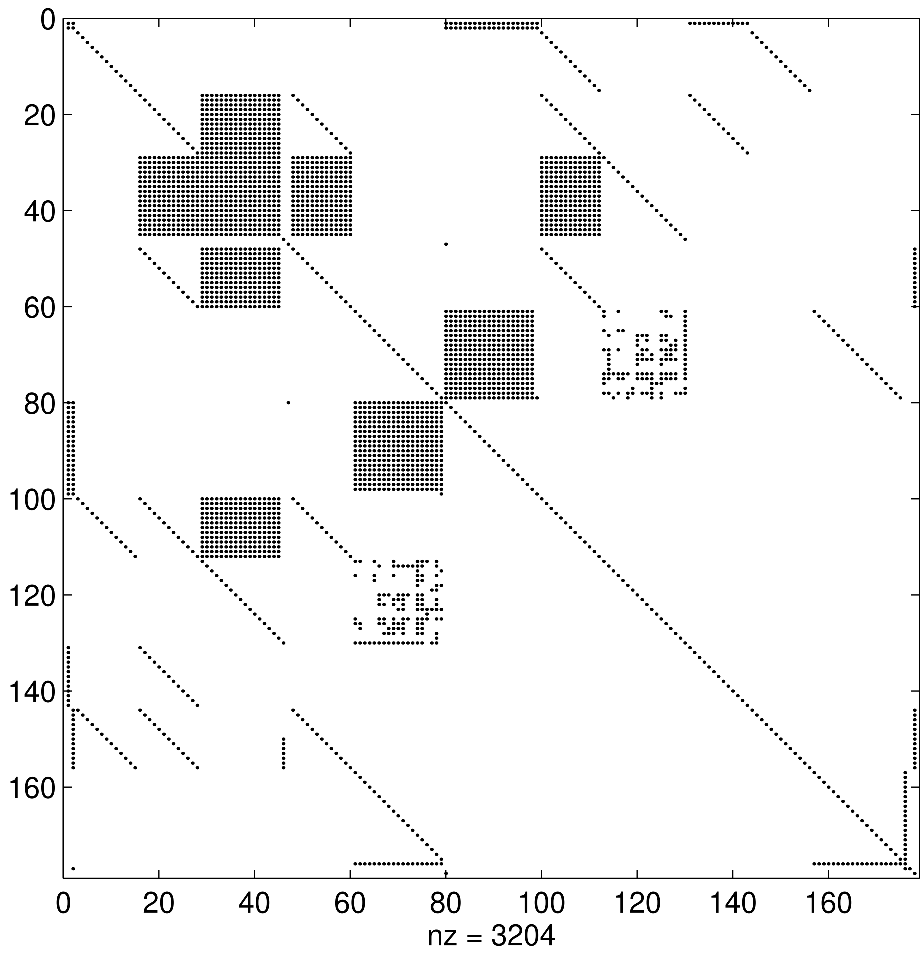

Figure 1: Jacobian of model gemmge

The matrix `J` is a MATLAB sparse matrix containing the evaluation of $\nabla f(z)$. Note that the (dense) vector `f` is output at the same time containing an evaluation of $f(z)$. This vector should be identical to that given by `cpfun`.

At this point, any of the MATLAB utilities can be used to formulate an algorithm, inspect the sparsity pattern of the Jacobian or modify the current iterate. We now give three examples of the use of the interface.

In the first example, we show how the interface may be effective for visualization of problem structure. This in turn may lead to improved modeling, or to algorithms that can exploit the underlying structure of many complementarity problems.

```
matlab> global wsvec cpfname
matlab> [z,l,u] = cpsetup('gemmge');
matlab> [f,J] = cpjac(z);
matlab> spy(J);
```

The resulting plot of the Jacobian of this model is given in Figure 1. The postscript file used here was generated after the "spy" command, by invoking "print -dps gemmge". At this stage, we can demonstrate some of the potentially useful features MATLAB for algorithmic design. For large scale problems, many algorithms are limited by their ability to solve large sparse systems of equations. Frequently, reordering of the matrices improves sparsity of the factors without creating numerical difficulties. MATLAB has several popular orderings included in its toolbox, for example the minimum degree ordering and the reverse Cuthill-McKee ordering. The resulting plots from applying

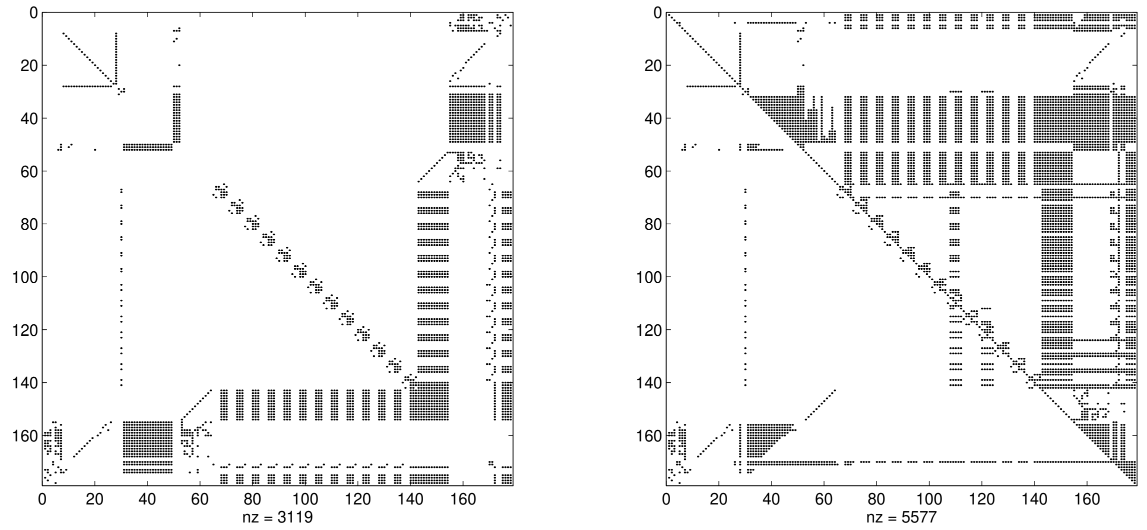

Figure 2: Jacobian reordered by Minimum Degree and resulting sparsity of L+U

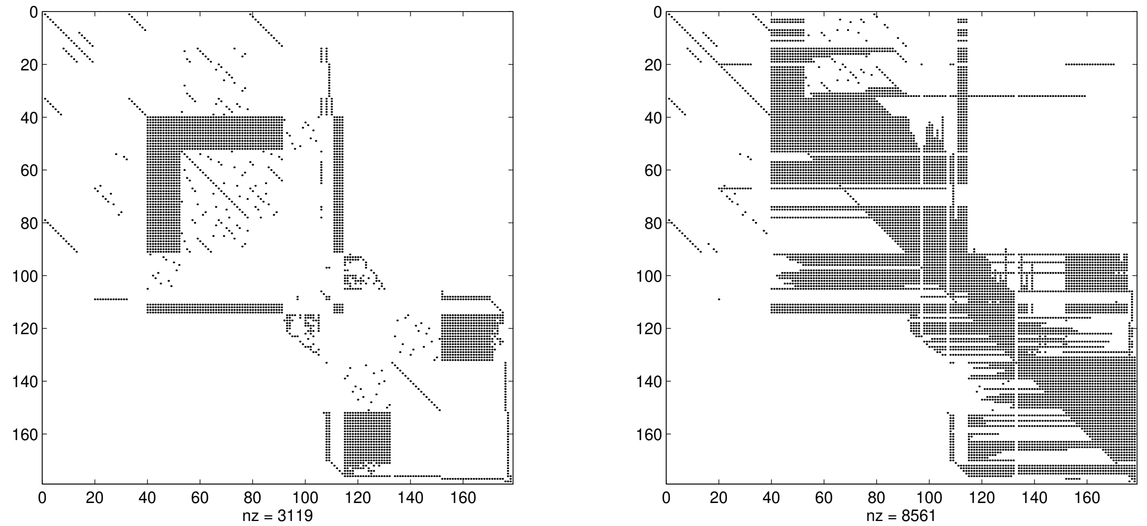

Figure 3: Jacobian reordered by Reverse Cuthill-McKee and resulting sparsity of L+U

these orderings to the "gemmge" Jacobian are given in Figures 2 and 3, along with a plot showing the resulting fill in the LU factors. Minimum degree clearly outperforms reverse Cuthill-McKee on this problem.

The second and third examples are matlab "m" files that use the interface routines to implement a simple projected gradient and projected Newton solver for complementarity problems. The first step in generating these algorithms is to set up a merit function. The file displayed in Figure 4, "merit.m" evaluates

$$\sqrt{\sum_{i:z_i<=l_i}(f_i(z))_-^2+\sum_{i:z_i>=u_i}(f_i(z))_+^2+\sum_{i:l_i<z_i<u_i}f_i(z)^2}$$

assuming it is given the current values of z, l, u and $f(z)$. The subscripts $+$ and $-$ correspond to the positive and negative parts of f respectively. Note that the global variables `wsvec` and `cpfname` are not needed within this routine. Once this function has been implemented, the code in Figure 5 can be used to set up a simple projected gradient algorithm for solving the MCP. These matlab codes can be invoked at the MATLAB command line as follows.

```
matlab> global wsvec cpfname
```

```
function norm = merit (z, f, l, u)

norm = 0;
for i=1:length(z)
    if (z(i) <= l(i))
      dt = min(0,f(i)); norm = norm + dt*dt;
    elseif (z(i) >= u(i))
      dt = max(0,f(i)); norm = norm + dt*dt;
    else
      norm = norm + f(i)*f(i);
    end
end
norm = sqrt(norm);
```

Figure 4: MATLAB code to evaluate merit function

```
function z = pgrad(z,l,u)
global wsvec cpfname
step_tol = 1e-2;

[f,J] = cpjac(z); residual = merit(z,f,l,u),

for iter=1:10
  alpha = 1.0;
  while (alpha > step_tol)
    znew = min(u,max(l,z - alpha*f));
    fnew = cpfun(znew); resnew = merit(znew,fnew,l,u);
    if (resnew < residual)
      break; end
    alpha = alpha * 0.5;
  end

  if (alpha <= step_tol)
    disp('linesearch failure'); return; end

  z = znew; f = fnew; residual = resnew,
  if (residual < 1e-6)
    return; end
end;
```

Figure 5: MATLAB code for projected gradient algorithm

```
matlab> [z,l,u] = cpsetup('gemmge');
matlab> znew = pgrad(z,l,u);
```

A more sophisticated code is based on the projected Newton preprocessor implemented as part of PATH [8, 9]. A full description is given in [6], but the file in Figure 6, "pnewt.m" contains all of the main points. This algorithm can be invoked in the same manner as the pgrad algorithm, except that pgrad is replaced by pnewt. Note that it is easy to update the "m" file to test out the effectiveness of least squares approaches for solving the system of equations defining the search direction, or to implement proximal perturbations in the problem of finding the Newton direction.

These examples demonstrate the ease with which algorithms can be prototyped and tested using the MATLAB interface. Indeed, Tseng [16] has generated a suite of algorithms for complementarity problems which were easily modified to use this interface. However, the results using these codes are perhaps not illuminating, since all of these codes fail on a large subset of the models. It is currently unclear whether this is due to the simple implementation of the algorithms, or due to properties of the algorithms themselves. While it is easy to prototype algorithms on many problems using the MATLAB interface, it is important to realize that practical codes for complementarity problems, such as PATH [8], MILES [14], SMOOTH [4], NE/SQP [13], SEMISMOOTH [5] and QPCOMP [2] have many features added that make them much more effective than a simple MATLAB encoding would demonstrate. A detailed comparison of the above codes has been given in [1]. Discerning how to use the interface in the best manner possible remains the topic of further research.

3 PROBLEM CHARACTERISTICS

The current set of problems arise from two sources, the library of GAMS test problems and MCPLIB. The following two tables correspond to these two sources. Further documentation on the problems can be found in [15] and [7] respectively.

Note that the following abbreviations are used when referring to the type of the problem:

MCP	General mixed complementarity problem
MPSGE	General economic equilibrium from MPSGE
LCP	Linear complementarity problem
NCP	Nonlinear complementarity problem
LMCP	Linear mixed complementarity problem
NE	Nonlinear equations
NLP	Optimality conditions of a nonlinear program

4 FILE GENERATION & ADVANCED OPTIONS

The interface between GAMS/MCP and MATLAB is created using CPLIB [10] and a single binary file. The binary file is machine specific, being written using the I/O library facilities of GAMS. Many platforms are supported by this library. If the user has access to GAMS/MCP, it is possible to generate new binary files incorporating any GAMS/MCP model using the following procedure.

To generate a binary file, a GAMS/MCP model is created as described in [10]. A line is added to the `gms` file after the model specification that reads

```
function z = pnewt(z,l,u)
global wsvec cpfname
step_tol = 1e-2;

[f,J] = cpjac(z); residual = merit(z,f,l,u),

for iter=1:10
  basiscount = 0; needed = [];
  for i=1:length(z)
    if ~((z(i) <= l(i) & f(i) >= 0.) | (z(i) >= u(i) & f(i) <= 0.))
      basiscount = basiscount+1; needed(basiscount) = i;
    end
  end

  r = f(needed); B = J(needed,needed); d = B\r;
  dd = sparse(size(z)); dd(needed) = d;

  alpha = 1.0;
  while (alpha > step_tol)
    znew = min(u,max(l,z - alpha*dd));
    fnew = cpfun(znew); resnew = merit(znew,fnew,l,u);
    if (resnew < residual)
      break; end
    alpha = alpha * 0.5;
  end

  if (alpha <= step_tol)
    disp('linesearch failure'); return; end

  z = znew; residual = resnew,
  if (residual < 1e-6)
    return; end
  [f,J] = cpjac(z);
end;
```

Figure 6: MATLAB code for projected Newton algorithm

Model	Type	n	nnz	density
cafemge	MPSGE	101	900	8.82%
cammcp	NCP	242	1621	2.77%
cammge	MPSGE	128	1227	7.49%
cirimge	MPSGE	9	33	40.74%
co2mge	MPSGE	208	1463	3.38%
dmcmge	MPSGE	170	1594	5.52%
ers82mcp	MCP	232	1552	2.88%
etamge	MPSGE	114	848	6.53%
finmge	MPSGE	153	1915	8.18%
gemmcp	MCP	262	2793	4.07%
gemmge	MPSGE	178	3441	10.86%
hansmcp	NCP	43	398	21.53%
hansmge	MPSGE	43	503	27.20%
harkmcp	NCP	32	131	12.79%
harmge	MPSGE	11	60	49.59%
kehomge	MPSGE	9	75	92.59%
kormcp	MCP	78	423	6.95%
mr5mcp	NCP	350	1687	1.38%
nsmge	MPSGE	212	1408	3.13%
oligomcp	NCP	6	21	58.33%
sammge	MPSGE	23	117	22.12%
scarfmcp	NCP	18	150	46.30%
scarfmge	MPSGE	18	181	55.86%
shovmge	MPSGE	51	375	14.42%
threemge	MPSGE	9	77	95.06%
transmcp	LCP	11	34	28.10%
two3mcp	NCP	6	29	80.56%
unstmge	MPSGE	5	25	100.00%
vonthmcp	NCP	125	760	4.86%
vonthmge	MPSGE	80	594	9.28%
wallmcp	NE	6	25	69.44%

Table 1: GAMSLIB Models

Model	Type	n	nnz	density
bertsekas	NCP	15	74	32.89%
billups	NCP	1	1	100.00%
bert_oc	LMCP	5000	21991	0.09%
bratu	NLP	5625	33749	0.11%
choi	NCP	13	169	100.00%
colvdual	NLP	20	168	42.00%
colvnlp	NLP	15	113	50.22%
cycle	LCP	1	1	100.00%
ehl_k60	MCP	61	3721	100.00%
ehl_k80	MCP	81	6561	100.00%
ehl_kost	MCP	101	10201	100.00%
explcp	LCP	16	152	59.38%
freebert	MCP	15	74	32.89%
gafni	MCP	5	25	100.00%
hanskoop	NCP	14	129	65.82%
hydroc06	NE	29	222	26.40%
hydroc20	NE	99	838	8.55%
josephy	NCP	4	16	100.00%
kojshin	NCP	4	16	100.00%
mathinum	NCP	3	9	100.00%
mathisum	NCP	4	14	87.50%
methan08	NE	31	225	23.41%
nash	MCP	10	100	100.00%
obstacle	LMCP	2500	14999	0.24%
opt_cont	LMCP	288	4928	5.94%
pgvon105	NCP	105	796	7.22%
pgvon106	NCP	106	898	7.99%
pies	MCP	42	183	10.37%
powell	NLP	16	203	79.30%
powell_mcp	NCP	8	54	84.38%
scarfanum	NCP	13	98	57.99%
scarfasum	NCP	14	109	55.61%
scarfbnum	NCP	39	361	23.73%
scarfbsum	NCP	40	614	38.38%
sppe	NCP	27	110	15.09%
tobin	NCP	42	243	13.78%

Table 2: MCPLIB Models

```
option mcp = matlab;
solve josephy using mcp;
```

where `josephy` is the name of the model. A GAMS solver "matlab" is invoked at the solve instruction that dumps all the relevant information into a machine specific binary file, "matlab.bin" in the current directory. This file can then be renamed and used independently of GAMS.

We conclude this section with a complete syntactical description of the currently available `mex` and `m` files.

```
[z,l,u] = cpsetup('josephy');
```

The input argument is a string, which corresponds to a file "josephy.bin" that is located in the current directory. The three output arguments are required and contain starting values for z, along with the problem data l and u. Values of $\pm 10^{20}$ should be treated as infinite. This routine must be called before any other, and will write the vectors `wsvec` and `cpfname` into the matlab workspace. Typically, these vectors should be made global. The returned starting point is the first one encountered in the associated `gms` file.

```
z = cpstart('josephy',k);
```

This function returns the `k`th starting point for the problem named "josephy" from the `gms` file. An error occurs if the requested starting point is not found in this m-file.

```
[f,domerr] = cpfun(z);
```

The input argument z is a vector containing the point at which the function is to be evaluated. The function has a variable number of output arguments. The first output argument is required and is a vector of the same size as z containing an evaluation of $f(z)$. The second argument domerr is optional and contains the number of domain violations that occurred during the function evaluation. If the second output argument is omitted and a domain violation occurs, then a MATLAB error is generated.

```
[f,J,domerr] = cpjac(z);
```

The input argument z and first output argument f are identical to those of the function `cpfun`. The domerr output argument is optional and acts exactly as described above for `cpfun`. The required output J is an evaluation of $\nabla f(z)$ and is returned in MATLAB sparse matrix form.

```
\begin{verbatim}
[f,J,domerr] = cpfunjac(z,flag);
```

The first input argument z and first output argument f are identical to those of the function `cpfun`. The domerr output argument is optional and acts exactly as described above for `cpfun`. The second input argument $flag$ is 1 if J should contain an evaluation of $\nabla f(z)$ (which will be returned in MATLAB sparse matrix form), and is 0 if no such evaluation should be performed. Currently, this is implemented as an m-file that simply calls `cpfun` or `cpjac`.

```
t = cptype(v);
```

In conjunction with the `gms` files that were used to generate the binary files, it may sometimes be helpful to extract "name" information corresponding to variables in the model. The following function gives limited access to these names. Furthermore, for models that were originally generated as an MPSGE model [15], the output vector t holds variable type indicators, with the following meanings:

`t`(i)	=	1	production sector
		2	commodity price
		3	consumer income
		0	auxiliary variable (no structural information)

The input vector `v` is a mask indicating what names are required (`v`(i)=0 if that name is to be skipped). One use of this function is to mask out the auxiliary variables and corresponding equations from the Jacobian. The resulting matrix typically consists of a symmetric part and a skew symmetric part, corresponding to the production and consumer portions of the model.

5 CONCLUSIONS

The interface consists of a freely available suite of programs and binary files that make a substantial library of problems available within MATLAB. The data, function and Jacobian evaluations can be used independently of the GAMS/MCP program that was used to generate the binary problem files. However, the corresponding `gms` files provide a nice documentation of the underlying models.

The binary format means that the function and Jacobian evaluations can be executed quickly. Furthermore, many commercial enterprises may be willing to provide such binary files for algorithmic testing since it enables their actual data to remain anonymous. This could be crucial in enlarging the suite of available problems.

Clearly, the MATLAB interface is just one technique for accessing the data from the binary files. Future developments of the programs could make a similar Fortran or C interface available for these problems. It may also become necessary to allow the MATLAB user to access more of the data in the binary file from within MATLAB, when designing algorithms to better exploit structure. This needs to be determined after experimentation with the current interface.

References

[1] S. C. Billups, S. P. Dirkse, and M. C. Ferris. A comparison of algorithms for large scale mixed complementarity problems. Mathematical Programming Technical Report 95–16, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1995. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.

[2] S. C. Billups and M. C. Ferris. QPCOMP: A quadratic program based solver for mixed complementarity problems. Mathematical Programming Technical Report 95–09, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1995. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.

[3] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, CA, 1988.

[4] C. Chen and O. L. Mangasarian. A class of smoothing functions for nonlinear and mixed complementarity problems. *Computational Optimization and Applications, forthcoming*, 1995.

[5] T. De Luca, F. Facchinei, and C. Kanzow. A semismooth equation approach to the solution of nonlinear complementarity problems. Submitted to Mathematical Programming, 1995.

[6] S. P. Dirkse. *Robust Solution of Mixed Complementarity Problems.* PhD thesis, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1994. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.

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

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

[9] S. P. Dirkse and M. C. Ferris. A pathsearch damped Newton method for computing general equilibria. *Annals of Operations Research, forthcoming*, 1995.

[10] S. P. Dirkse, M. C. Ferris, P. V. Preckel, and T. Rutherford. The GAMS callable program library for variational and complementarity solvers. Mathematical Programming Technical Report 94-07, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1994. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.

[11] M. C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. Discussion Papers in Economics 95–4, Department of Economics, University of Colorado, Boulder, Colorado, 1995. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.

[12] MATLAB. *User's Guide.* The MathWorks, Inc., 1992.

[13] J. S. Pang and S. A. Gabriel. NE/SQP: A robust algorithm for the nonlinear complementarity problem. *Mathematical Programming*, 60:295–338, 1993.

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

[15] T. F. Rutherford. Extensions of GAMS for complementarity problems arising in applied economic analysis. *Journal of Economic Dynamics and Control, forthcoming*, 1995.

[16] P. Tseng. Matlab implementations of complementarity codes. Private Communication, 1995.