THE PDCGM 2.0 MANUAL

Jacek Gondzio *[1], Pablo González-Brevis[†1,2], and Pedro Munari[‡3]

[1]School of Mathematics, University of Edinburgh, Edinburgh, United Kingdom
[2]Engineering School, Universidad del Desarrollo, Concepción, Chile
[3]Production Engineering Department, Federal University of São Carlos, São Carlos-SP, Brazil

September 3, 2013

1 Introduction

This document contains general information about the PDCGM which is the acronym for the *primal-dual column generation method.* For a complete description of the method and for reference, please refer to [13, 14]. This software has been written on top of HOPDM (Higher Order Primal-Dual Method) [7, 8, 9] which is a state-of-the-art implementation of a primal-dual interior point algorithm [11, 25].

1.1 About the PDCGM

The PDCGM is a column generation method [20] based on an infeasible primal-dual interior point algorithm [11]. This method was originally proposed in [15] and further developed in [14]. It relies on sub-optimal and well-centred solutions (dual variables) of the restricted master problem (RMP) sent to the oracle. In order to obtain sub-optimal solutions, the tolerance used to solve each restricted master problem is dynamically adjusted starting from loose at the beginning and being gradually tightened once the column generation method gets close the the solution. The centrality feature of the method is provided by the use of a primal-dual interior point method (path-following method) which obtains points in the interior of the feasible set and in the proximity to the central path [11]. At the end of the process, the optimal solution of the master problem is obtained if such exists. In [14], the authors present theoretical properties as well as extensive computational evidence of using the PDCGM in solving relaxations of integer programming problems. Extensions to more general applications are discussed in [13]. In [21], the authors study how to combine the PDCGM in a branch-and-price framework to solve the vehicle routing problem with time windows. Since the method requires solving a series of closely related problems, it employs specially designed warmstarting techniques for interior point methods [10, 12].

Some features in the current version are:

▷ More than one column can be added to the RMP per iteration.

▷ The method can warmstart.

▷ Redundant columns can be removed.

2 Installation of the PDCGM

You can download the latest version[1] of the PDCGM at `http://www.maths.ed.ac.uk/~gondzio/software/pdcgm.html`. PDCGM/HOPDM is distributed as a compressed tar'ed and gzip'ed file `pdcgmDEMOv2.0.tar.gz`.

* `j.gondzio@ed.ac.uk`
† `pablogonzalez@ingenieros.udd.cl`
‡ `munari@dep.ufscar.br`
[1] The current version is 2.0

The implementation has been tested in different UBUNTU distributions with 32 and 64 bit architectures. Once you get the file `pdcgmDEMOv2.0.tar.gz`, open the Terminal window and type

```
> tar xvfz pdcgmDEMOv2.0.tar.gz
```

to create a new subdirectory `pdcgmDEMOv2.0` in the current directory. Once you have uncompressed the file, enter to `/pdcgmDEMOv2.0` and look at its content.

You will find the following subdirectories:

- ▷ `applications:` it contains a number of applications which use the PDCGM libraries;
- ▷ `data:` it contains instance files for some of the implementations in the `applications` directory;
- ▷ `extras:` it contains third-party files;
- ▷ `hopdm:` it contains the HOPDM library;
- ▷ `interface:` it contains a C interface to HOPDM library;
- ▷ `mkfhosts:` it contains examples of files which specify the names of compilers, paths to libraries, and a number of compilation options;
- ▷ `pdcgm:` it contains the PDCGM library;

and two files:

- ▷ `README.1st:` it is a short version of this manual.
- ▷ `MANUAL.pdf:` it is the manual for using the PDCGM that you are reading now.

To install PDCGM/HOPDM on UBUNTU, you will need the library *libf2c*. To install this library in your system, open a Terminal window and type

```
> sudo apt-get install libf2c2
> sudo apt-get install libf2c2-dev
> sudo rm /usr/lib/libf2c.so && sudo ln -s /usr/lib/libf2c.a /usr/lib/libf2c.so
```

Your `makefile` in the directory `mkfhosts` should have the line: *LIBS= -lm -lf2c* just below the tag *#Libraries:*. In all our applications we use `munari` located in `mkfhosts`... have a look!

3 Applications

We provide the current version of the PDCGM with source files for seven different applications. In the `applications` directory you will find the following subdirectories: `clspst, csp, demo_pdcgm, mcnf, mkl, tssp and vrptw`.

Some of the applications depend on third-party packages/libraries. For instance, `mkl` depends on the SHOGUN Machine Learning Toolbox [22][2] and `tssp` depends on the IBM CPLEX Optimizer package [16][3]. Please, make sure you have these packages installed and properly working on your machine before compiling and running the corresponding PDCGM applications.

For a full description of the applications, we point the reader to some references where more details about the problems and column generation formulations can be found. At this point, it is enough to say that for every application we rely on the Dantzig-Wolfe decomposition principle (DWD) [4] and that the resulting problem fits into the column generation framework [20].

In the `/applications` you can find the following directories:

- ▷ `clspst:` this directory contains the capacitated lot-sizing problem with setup times [17, 23] where the linking constraint is the capacity constraint. By using DWD, we gain separability per item and therefore the master problem is a disaggregated one (one column per item). The subproblem is a single-item lot-sizing problem without capacity constraint and we use the Wagner-Within algorithm [24] to solve it.

[2] `http://shogun-toolbox.org/`
[3] `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

- ▷ `csp:` this directory includes the cutting stock problem [2, 3]. The linking constraint is the demand constraint and since the width of the rolls is assumed to be the same, an aggregated master problem is obtained. The subproblem is a knapsack problem and more than one column per iteration may be obtained. We use a branch-and-bound method provided by Dr. Aline Leão [19] to solve the subproblem.

- ▷ `demo_pdcgm:` this directory contains three quadratic problems which can be solved by a cutting plane method. The problem uses Lagrangian relaxation and you can see the correspondence with the column generation method at every single file. Also, it provides some examples of how to feed the PDCGM with dense matrix representation.

- ▷ `mcnf:` this directory includes the multicommodity network flow problem [13]. In its standard linear programming formulation, the arc-node incidence matrix is replicated K times, where K is the number of commodities. These replicated matrices would be independent from each other, except for the capacity constraint. By taking this constraint as the linking constraint, the problem decomposes by commodity and a disaggregated master problem is obtained. The subproblem is a shortest path problem. An active set strategy is used as in [1] to determine which capacity constraints may be relaxed without compromising optimality.

- ▷ `mkl:` this directory contains the multiple kernel learning problem [13] formulated as a quadratically constrained quadratic problem (QCQP). By applying DWD for general convex programs [6], we obtain an aggregated master problem. The subproblem is a single kernel problem which is solved using the SHOGUN Machine Learning Toolbox [22].

- ▷ `tssp:` this directory includes the two-stage stochastic programming problem [13]. We solve the dual of the deterministic equivalent problem [26]. The linking constraints are the ones which relate the first-stage variables in the dual problem. The problem decomposes by scenario, so for each scenario we have a subproblem which is the dual of the second-stage problem. The subproblems are solved using the optimization package IBM ILOG CPLEX [16]. For this application we have an aggregated master problem and the subproblems may generate extreme points as well as extreme rays.

- ▷ `vrptw:` this directory contains the vehicle routing problem with time windows [5, 18]. The linking constraint is the demand constraint (every customer has to be visited) and since the vehicles are assumed to be identical, an aggregated master is obtained. The subproblem is an elementary shortest path problem with resource constraints. We solve a relaxed version of this subproblem in which non-elementary paths are allowed (i.e., paths that visit the same customer more than once).

In each of these directories you will find:

- ▷ `<application name>-pdcgm:` it is the source file of the application;

- ▷ `example:` it is a file with a toy example for the application;

- ▷ `include:` it is the directory that includes some useful libraries such as the subproblem solver;

- ▷ `makefile:` it compiles the source code and creates the executable;

- ▷ `run-instances:` it is a script that runs all the instances included in the `/data` directory.

3.1 Compiling and using the applications

In our experience a good strategy would be to try the applications in `clspst, csp, demo_pdcgm, mcnf` and/or `vrptw` directories first since these are stand-alone applications. If everything is working fine with these applications, you could then try with the `mkl` and `tssp` which depend on third-party packages. Before running any of the applications, we suggest you to read the `README.1st` file included in every application subdirectory.

3.1.1 CLSPST

To compile and run the CLSPST application, go to `/applications/clspst` and type

```
> make
> ./clspst-pdcgm <instance file>
```

For each CLSPST instance solved successfully, the following statistics are printed in the file `output-clspst-pdcgm.txt`

Instance - Lower bound - Upper bound - Relative gap - Outer iterations - RMP time (s) - Oracle time (s) - Total time (s)

There is an instance example in this directory which you can run by typing

```
> ./clspst-pdcgm example.txt
```

You will find more CLSPST instances in `/data/clspst`. In case you wish to run the PDCGM on all those instances, you can type

```
> ./run-instances
```

3.1.2 CSP

To compile and run the CSP application, go to `/applications/csp` and type

```
> make
> ./csp-pdcgm <instance file> <max number of cols>
```

If `<max number of cols>` is omitted, then it sets `<max number of cols> = 100`. For each CSP instance solved successfully, the following statistics are printed in the file `output-csp-pdcgm.txt`

Instance - Lower bound - Upper bound - Relative gap - Outer iterations - RMP time (s) - Oracle time (s) - Total time (s)

There is an instance example in this directory which you can run by typing

```
> ./csp-pdcgm example.txt
```

You will find more CSP instances in `/data/csp`. In case you wish to run the PDCGM on all those instances, you can type

```
> ./run-instances
```

3.1.3 DEMO_PDCGM

To compile and run the `qpexample1` inside `demo_pdcgm`, go to `/applications/demo_pdcgm` and type

```
> cp qpexample1.c example-pdcgm.c
> make
> ./example-pdcgm.c
```

Similar statements can be used to run `qpexample2` and `qpexample3`. A full description of the quadratic problem and the column generation implementation is available at the beginning of each file.

3.1.4 MCNF

To compile and run the MCNF application, go to `/applications/mcnf` and type

```
> make
> ./mcnf-pdcgm <instance_name> <p1> <p2>
```

where <instance_name> should be replaced by the full path to the input file; <p1> should be 1 for an aggregated model or 0, otherwise; <p2> should be 1 for using the active set strategy or 0, otherwise. In case <p1> and <p2> are both omitted, it assumes <p1>=0 and <p2>=1.

For each MCNF instance solved successfully, the following statistics are printed in the file output-mcnf-pdcgm.txt

Instance - Lower bound - Upper bound - Relative gap - % of active sets - Outer iterations - RMP time (s) - Oracle time (s) - Total time (s)

There is an instance example in this directory and you can run it by typing

```
> ./mcnf-pdcgm example.dat 0 1
```

You will find more MCNF instances in /data/mcnf. In case you wish to run the PDCGM on all those instances, you can type

```
 > ./run-instances
```

This will solve the instances using the default settings (<p1>=0 and <p2>=1).

3.1.5 MKL

Before using this code, you must have installed on your machine the *g++* compiler and the package SHOGUN

SHOGUN setup. In order to compile and run the MKL application, you must have the SHOGUN Machine Learning Toolbox [22] installed on your machine. We recommend you to use the version available in /extras of this PDCGM distribution. First, extract the file shogun-toolbox.tar.gz. Then, open the terminal console, go to the generated directory and type

```
> cd src
> ./configure
> make
> sudo make install
```

After these steps, SHOGUN should be working fine. In case something goes wrong, we suggest you have a look at the files INSTALL and README available on the directory shogun-toolbox/src. For further information, go to http://shogun-toolbox.org.
After installing SHOGUN, you will be able to run the MKL. To compile and run the MKL application, go to /applications/mkl and type

```
> make
> ./mkl-pdcgm <instance_name>
```

There is an instance example in this directory which you can run by typing

```
> ./mkl-pdcgm example.txt
```

For each MKL instance solved, the following statistics are printed in the file output-mkl-pdcgm.txt

Instance - Lower bound - Upper bound - Relative gap - Outer iterations - RMP time (s) - Oracle time (s) - Total time (s) - Number of kernels - Accuracy

You will find more MKL instances in /data/mkl. In case you wish to run PDCGM on all those instances, you can type

```
 > ./run-instances
```

3.1.6 TSSP

In order to compile this application, you must have CPLEX [16] installed on your machine. Please, check in the `makefile` available in the current directory if the flags `CPLEXDIR`, `SYSTEM` and `LIBFORMAT` are properly set according to your CPLEX installation. Certain versions of CPLEX require the library *ia32-libs* to be installed. If this is the case, you can install this library by typing on the terminal

```
> sudo apt-get install ia32-libs
```

To compile and run the TSSP application, go to `/applications/tssp` and type

```
> make
> ./tssp-pdcgm <instance file.cor> <instance file.tim> <instance file.sto>
```

By default, this setting allows an aggregated formulation. Additionally, you can specify if a disaggregated formulation should be used by typing

```
> ./tssp-pdcgm <instance file.cor> <instance file.tim> <instance file.sto> 0
```

For each TSSP instance solved, the following statistics are printed in the file `output-tssp-pdcgm.txt`

Instance - Number of scenarios - Lower bound - Upper bound - Relative gap - Outer iterations - RMP time (s) - Oracle time (s) - Total time (s)

There is an instance example in this directory which you can run by typing

```
> ./tssp-pdcgm example.cor example.tim example.sto
```

You will find more TSSP instances in `/data/tssp`. In case you wish to run the PDCGM on all those instances, you can type

```
> ./run-instances
```

3.1.7 VRPTW

To compile and run the VRPTW application, go to `/applications/vrtpw` and type

```
> make
> ./vrptw-pdcgm <instance file> <max number of cols>
```

If `<max number of cols>` is omitted, then it sets `<max number of cols> = 100`. For each VRPTW instance solved, the following statistics are printed in the file `output-vrptw-pdcgm.txt`

Instance - Lower bound - Upper bound - Relative gap - Outer iterations - RMP time (s) - Oracle time (s) - Total time (s)

There is an instance example in this directory which you can run by typing

```
> ./vrptw-pdcgm example.txt
```

You will find more VRPTW instances in `/data/vrptw`. In case you wish to run PDCGM on all those instances, you can type

```
> ./run-instances
```

4 Developing a new application

When developing an application on top of PDCGM, the user must define the two main components of a column generation procedure: (*i*) the master problem; (*ii*) the oracle function. We advise the user to start developing the master problem structure and then set the PDCGM environment with all this information. In the applications provided in `pdcgmDEMOv.2.0`, this task is done in the `main` function of the C/C++ files. Here is an example with the basic statements of a `main` function, where `k` and `v` are the number of linking constraints and the number of convexity constraints in the master problem, respectively.

4.1 Main function

```
Set the PDCGM environment
PDCGM *PDCGM_env

Allocate memory for the master problem
b = PDCGM_ALLOC(double, k + v + 1) → right hand side vector
u0 = PDCGM_ALLOC(double, k + v + 1) → initial guess for the duals (can be NULL)
row_type = PDCGM_ALLOC(double, k + v + 1) → constraints type (=, ≤, ≥ or objective)
lo_box = PDCGM_ALLOC(double, k) → lower bound of each dual variable
up_box = PDCGM_ALLOC(double, k) → upper bound of each dual variable

Populate the internal data structure of the PDCGM environment
PDCGM_env = PDCGM_set_data(
 k, → number of linking constraints in the RMP
 v, → number of convexity constraints in the RMP
 (k + v + 1), → maximum number of nonzeros in a column
 max_ncols_oracle, → maximum number of columns generated in a call to the oracle
 max_outer, → maximum number of outer iterations
 u0, → initial guess of the dual solution (can be NULL)
 b, → RHS of each constraint in the RMP
 row_type, → type of each constraint (row) in the RMP
 lo_box, → lower bound vector of the dual variables in the RMP
 up_box, → upper bound vector of the dual variables in the RMP
 instance) → a pointer to the instance data (any data structure defined by the user)

Set parameter δ: optimality tolerance for the column generation algorithm
PDCGM_set_delta(PDCGM_env, optimality_tolerance)

Set parameter D: degree of optimality. It must be greater than 1.0
PDCGM_set_degree_of_optimality(PDCGM_env, D)

Set parameter ε_max: maximum optimality tolerance used to solve the RMP
PDCGM_set_max_opt_tol(PDCGM_env, epsilon_max)

Set parameter: verbose mode (how much information is printed)
0: only info about HOPDM; 1: info about PDCGM; 2: more info about PDCGM
PDCGM_set_verbosity(PDCGM_env, 1)

Start the column generation procedure (the oracle function must be sent as a parameter)
PDCGM_solve_MP(PDCGM_env, oracle)
```

The entries in `row_type` must be defined by using the following macros

`EQUAL`, `LESS_EQUAL`, `GREATER_EQUAL` or `OBJECTIVE`.

We need to draw the reader's attention to two important structures in the previous example. First, is the variable `instance` which is sent as a parameter in function `PDCGM_set_data()`. It should be a pointer to the data structure defined by the user, but it can be sent as `NULL`, in case no data structure is needed in the application. This pointer is sent to oracle function, every time PDCGM calls this function, in order to have the instance data available in the oracle.

The second important structure in the example is the `oracle` variable, which must be a pointer to a function that performs all the tasks required in the oracle (*e.g.*, calling a pricing subproblem, creating a matrix of generated columns, adding columns to the RMP). As mentioned above, this function is one of the main components to be defined in a column generation procedure. At each outer iteration, PDCGM solves the restricted master and then calls the oracle function. An example of an oracle function with the main statements is given in the next section.

4.2 Oracle function

```
static short oracle(
 double *primal_violation, → violation of the generated constraints (returning parameter)
 double *dual_violation, → relative cost of the generated columns (returning parameter)
 PDCGM *PDCGM_env, → a pointer to the PDCGM environment
 void *instance_data) → a pointer to the instance data
{
 Get the pointer to the dual solution of the current RMP
 double *u = PDCGM_get_dual_solution(PDCGM_env)

 Set a sparse matrix that will be used to store the generated columns
 PDCGM_SMatrix_CW *M

 Allocate the sparse matrix in memory
 M = PDCGM_ALLOC(PDCGM_SMatrix_CW, 1)
 PDCGM_set_SMatrix_CW(M, max_nrows, max_ncols, max_nnz)

 At this point, the pricing subproblem should be called. Then, the solution provided by the
subproblem should be used to generate columns (if possible). These columns must be written
in the sparse matrix

 Add the generated columns to the RMP
 PDCGM_add_columns(PDCGM_env, M, NULL)

 Set the relative cost of the generated columns
 *dual_violation = subproblem_value

 Free up the allocated memory
 PDCGM_free_SMatrix_CW(M)
 PDCGM_FREE(M)

 Standard return in column generation
 return 1
}
```

The values `max_nrows`, `max_ncols` and `max_nz` refer to the maximum number of rows, the maximum number of columns and the maximum number of nonzero entries which must be allocated in `M`. A new column in PDCGM (aggregated version) must have the coefficients in the form $[\overline{A}_j, 1, \overline{c}_j]^T$, where $\overline{A}_j$ is the array of coefficients in the linking constraints, and $\overline{c}_j$ is the cost coefficient of the column. For a description of how to use the sparse matrix representation, see the APPENDIX A at the end of this document. A dense representation may also be used in PDCGM. For an example of this, please see the source codes in the directory `/applications/demo_pdcgm`. We recommend using the dense representation only in a preliminary implementation, as it may slow down the performance of the column generation procedure.

Notice that the parameters `primal_violation` and `dual_violation` must be set inside the oracle. The parameter `primal_violation` is intended for row generation and, hence, it is not used in a pure column generation procedure (the row generation is not covered in this manual). The (best) relative cost of the generated columns should be assigned to `dual_violation`. The oracle should always return 1, in case of column generation (even though no columns are generated).

In summary, the oracle function must: (i) call the pricing subproblem in order to generate one or more columns; (ii) in case new columns were generated, set them in an auxiliary matrix; (iii) add the auxiliary matrix to the RMP; and (iv) set the relative cost. It is worth mentioning that a similar function can be defined by the user, in order to set initial columns to the RMP. This function must be called once in the main function (the one used to set the master problem). An example of this operation is implemented in the applications provided with the `pdcgmDEMOv2.0`.

5 Final remarks

We provide the PDCGM code with no warranty, so be aware of using it at your own risk. We are not able to provide much support for the code, but we would appreciate if you could report any bug you may find when using the code. We would be happy to hear about your experience while using the code, specially about the applications you implement and the results you obtain. This code must be used for academic purposes only and should never be redistributed. Please, refer to one of the authors or `http://www.maths.ed.ac.uk/~gondzio/software/pdcgm.html` if someone is interested in using it. When reporting the use of the code, please cite [13, 14].

We wish you a lot of fun when using the PDCGM/HOPDM code.

Best regards,

Jacek, Pablo and Pedro

References

[1] Babonneau, F., du Merle, O., Vial, J.P.: Solving large-scale linear multicommodity flow problems with an active set strategy and proximal-ACCPM. Operations Research **54**(1), 184–197 (2006)

[2] Ben Amor, H., Valério de Carvalho, J.: Cutting stock problems. In: G. Desaulniers, J. Desrosiers, M.M. Solomon (eds.) Column Generation, pp. 131–161. Springer US (2005)

[3] Briant, O., Lemaréchal, C., Meurdesoif, P., Michel, S., Perrot, N., Vanderbeck, F.: Comparison of bundle and classical column generation. Mathematical Programming **113**, 299–344 (2008)

[4] Dantzig, G.B., Wolfe, P.: The decomposition algorithm for linear programs. Econometrica **29**(4), 767–778 (1961)

[5] Desrochers, M., Desrosiers, J., Solomon, M.: A new optimization algorithm for the vehicle routing problem with time windows. Operations Research **40**(2), 342–354 (1992)

[6] Geoffrion, A.M.: Elements of large-scale mathematical programming Part II: Synthesis of algorithms and bibliography. Management Science **16**(11), 676–691 (1970)

[7] Gondzio, J.: HOPDM (version 2.12) - a fast LP solver based on a primal-dual interior point method. European Journal of Operational Research **85**, 221–225 (1995)

[8] Gondzio, J.: Multiple centrality corrections in a primal-dual method for linear programming. Computational Optimization and Applications **6**(2), 137–156 (1996)

[9] Gondzio, J.: Presolve analysis of linear programs prior to applying an interior point method. INFORMS Journal on Computing **9**(1), 73–91 (1997)

[10] Gondzio, J.: Warm start of the primal-dual method applied in the cutting-plane scheme. Mathematical Programming **83**, 125–143 (1998)

[11] Gondzio, J.: Interior point methods 25 years later. European Journal of Operational Research **218**(3), 587–601 (2012)

[12] Gondzio, J., González-Brevis, P.: A new warmstarting strategy for the primal-dual column generation method. Technical Report, ERGO 12-007, University of Edinburgh, School of Mathematics (2012)

[13] Gondzio, J., González-Brevis, P., Munari, P.: Large-scale optimization with the primal-dual column generation method. Technical Report, ERGO 13-013, University of Edinburgh, School of Mathematics (2013)

[14] Gondzio, J., González-Brevis, P., Munari, P.: New developments in the primal-dual column generation technique. European Journal of Operational Research **224**(1), 41–51 (2013)

[15] Gondzio, J., Sarkissian, R.: Column generation with a primal-dual method. Technical Report 96.6, Logilab (1996)

[16] IBM ILOG CPLEX v.12.1: Using the CPLEX Callable Library (2010)

[17] Jans, R., Degraeve, Z.: Improved lower bounds for the capacitated lot sizing problem with setup times. Operations Research Letters **32**(2), 185 – 195 (2004)

[18] Kallehauge, B., Larsen, J., Madsen, O.B., Solomon, M.M.: Vehicle routing problem with time windows. In: G. Desaulniers, J. Desrosiers, M.M. Solomon (eds.) Column Generation, pp. 67–98. Springer US (2005)

[19] Leão, A.A.S.: Geração de colunas para problemas de corte em duas fases. Master's thesis, ICMC - University of Sao Paulo, Brazil (2009)

[20] Lübbecke, M.E., Desrosiers, J.: Selected topics in column generation. Operations Research **53**(6), 1007–1023 (2005)

[21] Munari, P., Gondzio, J.: Using the primal-dual interior point algorithm within the branch-price-and-cut method. Computers & Operations Research **40**(8), 2026–2036 (2013)

[22] Sonnenburg, S., Rätsch, G., Henschel, S., Widmer, C., Behr, J., Zien, A., de Bona, F., Binder, A., Gehl, C., Franc, V.: The shogun machine learning toolbox. Journal of Machine Learning Research **11**, 1799–1802 (2010)

[23] Trigeiro, W.W., Thomas, L.J., McClain, J.O.: Capacitated lot sizing with setup times. Management Science **35**(3), 353–366 (1989)

[24] Wagner, H.M., Whitin, T.M.: Dynamic version of the economic lot size model. Management Science **5**(1), 89–96 (1958)

[25] Wright, S.J.: Primal-Dual Interior-Point Methods. SIAM (1997)

[26] Zverovich, V., Fábián, C.I., Ellison, E.F., Mitra, G.: A computational study of a solver system for processing two-stage stochastic LPs with enhanced Benders decomposition. Mathematical Programming Computation **4**, 211–238 (2012)

A Sparse representation

To fill in the matrix `M`, you may use sparse or dense matrix representations. For details of dense representation, see `/applications/demo_pdcgm` and the examples therein. The basic idea of a sparse representation is to assign some pointers and value to the nonzero elements in a matrix. Note that for the sparse representation, FORTRAN indexing must be used (starting from 1 and not from 0). There are three vectors that describe a matrix.

- ▷ `clpnts`: vector of column pointers;
- ▷ `coeff`: vector of coefficients;
- ▷ `rwnmbs`: vector with row numbers.

For instance, assume we have a matrix

$$A = \begin{vmatrix} 0 & 3 & 2 & 0 & 1 \\ 2 & 4 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 3 & 2 & 4 & 1 & 5 \end{vmatrix}$$

The matrix dimensions and number of nonzero elements is given as

```
M->m = 4;  /*number of rows*/
M->n = 5;  /*number of columns*/
M->nz= 15; /*number of nonzero elements*/
```

The elements of the matrix have to be given in a column-wise fashion. `clpnts[i]` denotes with which element the $(i+1)$-th column starts with. We always start with `M->clpnts[0] = 1`. In our toy example, matrix A has three nonzero elements in the first column and therefore, the counter is updated from 1 to 4. The second column has four nonzero elements so the counter is updated from 4 to 8... and so on. The `clpnts` vector for this example is

```
M->clpnts[0] = 1;
M->clpnts[1] = 4;
M->clpnts[2] = 8;
M->clpnts[3] = 11;
M->clpnts[4] = 13;
M->clpnts[5] = 15;
```

The nonzero elements of the matrix are added in the following way

```
Element in row 2, column 1: 2
M->coeff[1] = 2;
M->rwnmbs[1] = 2;

Element in row 3, column 1: 1
M->coeff[2] = 1;
M->rwnmbs[2] = 3;
....

Element in row 3, column 3: 1
M->coeff[9] = 1;
M->rwnmbs[9] = 3;
....

Element in row 4, column 5: 5
M->coeff[15] = 5;
M->rwnmbs[15] = 4;
```