Polynomial Moment Optimization Database

B. Mourrain, M. Kocvara

6 July 2020

This document describes the specification of the database of polynomial & moment optimization problems, developed in the network POEMA.

The main features of this database are the following:

- The data are Polynomial Moment Optimization problems (PMO) described in a certain format (Section Format) in files or ressources freely accessible and characterized by a Universally Unique IDentifier (Section UUID).
- Datasets collecting all the data available in the database and metadata are provided.
- Data can be accessible from everywhere via their UUID (Section UUID).
- Data sets can be retrieved and used remotely, from applications and environments such as Julia (Section DataBase access)
- Members of the network and collaborators can upload data on the server (Section DataBase access)

Format for PMO data

The format used to encode the data is a numeric ascii `json` based format. An optimization problem is composed of a list of variables, an objective function and constraints:

- `"name":` (optional) the name of the problem.
- `"variables":` (optional) a list of strings `["x","y",...]` for the names of the variables.
- `"vartypes":` (optional) a list of strings specifying the type of the variables.
- `"objective":` a function constraint (see below).
- `"constraints":` a list of function constraints (see below).
- `"author":` (optional) a string with the name(s) of the author(s), who produced it.
- `"version":` (optional) the version of the data as a string.
- `"uuid":` a string identifier (see Section UUID)

We consider three types of PMO data:

- Polynomial data
- Moment data
- SDP data

Polynomial data

The polynomial optimization problems are problems of the form:

$$\begin{array}{ll} \inf\limits_{\mathbf{x}\in\mathbb{R}^n} & f(\mathbf{x}) \\ s.t. & g_1(\mathbf{x}) \geq 0, \ldots, g_{n_1}(\mathbf{x}) \geq 0 \\ & h_1(\mathbf{x}) = 0, \ldots, h_{n_2}(\mathbf{x}) = 0 \end{array}$$

where f, g_i, h_j are polynomials in the variables $x = (x_1, \ldots, x_n)$ with real coefficients in R. The function f is the objective function, g_i are sign constraints and h_j are equality constraints. By changing f in $-f$ or g_i in $-g_i$ we can replace inf by supor non-negativity constraints by non-positivity constraints.

This family of problems includes

- global polynomial optimization with no constraint ($n_1 = n_2 = 0$),
- polynomial system solving when $f \equiv 0$ and $n_1 = 0$.

For polynomial data, the `"type"` attribute is :

- `"type": "polynomial"`

The `"objective"` attribute is a component with the following attributes.

- `"set":` with value in `{"inf", "sup"}`. If it is an objective function to be minimized or maximized, the value is in `{"inf", "sup"}`.
- `"polynomial":` a polynomial.

Similarly, the `"constraints"` attribute is components `{"set": ..., "polynomial": ,...}` with

- `"set":` value in `{"=0", "<=0", ">=0"}`.
- `"polynomial":` a polynomial.

A polynomial $p = \sum_{i=1}^{l} p_{\alpha_i} x^{\alpha_i}$ where $p_{\alpha_i} \in R$ and $\alpha_i = (\alpha_{i,1}, \ldots, \alpha_{i,n}) \in N^n$ is represented in its sparse representation in the monomial basis or any specified basis, with the following attributes:

- `"terms":` with value a list of terms encoded as `[c, [d1, …], [v1, …]]` where c is the coefficient, `[d1, …]` the list of exponents for the variables of indices `[v1, …]`.
- `"nvar":` with value the number of variables.
- `"nterm":` (optional) with value the number of terms.
- `"coeftype":` (optional) with value the type of the coefficients: `{"Int64", "Float64", p, …}` where `p` is an integer which stands for Z/pZ
- ...

Here is the encoding for the polynomial constraint $x^4 - 4xy^3 + y^2 - 2 = 0$:

```
{
  "set": "=0",
  "polynomial": {
     "nvar": 2,
     "coeftype": "Int64",
     "terms": [
     [1,[4],[1]],
     [-4,[1,3]],
     [1,[2],[2]],
     [-2]
     ]
  }
}
```

The first term has one exponent 4 in the first variable, which is x. The second term, with coefficients -4, has exponents [1,3] in the variables [x,y]. Since all the variables appear, the indices of the variables are not indicated. The third term y^2 involves only the second variables with exponent 2. The constant coefficient is the fourth term, with coefficient -2 and no exponen and no variable.

Here is the encoding of the Polynomial Optimization problem

$$inf\ x^4 + x^2y^2 - y^3\ s.t. \qquad x^2 + 3.14159265y^2 - 2 \leq 0$$

$$x \in [-1,1]$$

$$2y^2 - y = 0$$

```
{
  "type": "polynomial",
  "variables": [
     "x",
     "y"
  ],
  "nvar": 2,
  "objective": {
     "set": "inf",
     "polynomial": {
          "coeftype": "Float64",
          "terms": [
              [1.0,[4],[1]],
              [1.0,[2,2]],
              [-1.0,[3],[2]]
          ]
```

```
      }
  },
  "constraints": [
      {
            "set": "<=0",
            "polynomial": {
                  "coeftype": "Float64",
                  "terms": [
                        [1.0,[2],[1]],
                        [3.141592653589793,[2],[2]],
                        [-2.0]
                   ]
            }
      },
      {
            "set": [-1,1],
            "polynomial": {
                  "coeftype": "Int64",
                  "terms": [
                        [1,[1],[1]]
                  ]
            }
      },
      {
            "set": "=0",
            "polynomial": {
                  "coeftype": "Float64",
                  "terms": [
                        [2.0,[2],[2]],
                        [-1.0,[1],[2]]
                  ]
            }
      }
  ],
  "version": "0.0.1",
  "author": "Bernard Mourrain”,
  "uuid": "82d25714-9513-11ea-22da-9dbe2a261539"
}
```

Moment Optimization

Optimization problems on moment sequences can also be described:

$inf\ \Sigma_i\ \langle f_i * \mu_i, 1\rangle$ s.t.

$$\Sigma_i\ g_i * \mu_i \succcurlyeq 0, \ldots$$

$$\Sigma_i\ h_i * \mu_i = 0, \ldots$$

$$\Sigma_i\ \langle p_i * \mu_i, 1\rangle\ + c_0\ \geq\ 0, \ldots$$

$$\Sigma_i\ \langle q_i * \mu_i, 1\rangle\ + d_0\ =\ 0, \ldots$$

The type of the variables are moment sequences, the constraints are linear moment sequence equalities or inequalities. Here is an example:

$inf\ \ \langle \mu_1, x^4 + x^2y^2 - y^3\rangle\ st.$

$$(x^2\ +\ 3.14159265y^2\ -\ 2) * \mu_1 +\ (2\ y^2\ -y) * \mu_2\ \succcurlyeq 0$$

$$\langle x^2 * \mu_1, 1\ \rangle\ -\ \langle y * \mu_2, 1\ \rangle\ -\ 3\ \geq\ 0$$

For moment data, the `"type"` attribute is :

- `"type": "moment"`

The `"objective"` attribute is a component with the following attributes.

- `"set":` with value in `{"inf", "sup"}`. If it is an objective function to be minimized or maximized, the value is in `{"inf", "sup"}`.
- `"moments":` a vector of polynomial multipliers.

Similarly, the `"constraints"` attribute is a list of components `[{"set": ..., "moments": ,...}...]` where

- `"set":` with value in `{"=0", "<=0", ">=0", "=0 *", "<=0 *", ">=0 *"}` to specify that either the functional is positive, negative or 0 or the moment value is positive, negative or 0. If the string ends with a `*`, it indicates it is a mass constraint.
- `"moments":` a sequence of terms representing the expressions $\Sigma_i\ h_i * \mu_i$ where
  - `"coeftype":` is the type of the coefficients of the polynomials
  - `"terms":` is the list of terms represented as for polynomial optimization problems, except that there is an additional index. This first index is the index of the moment sequence μ_i in the expression $\Sigma_i\ h_i * \mu_i$ and the other values represent respectively the coefficients of the term in , its exponent vector, and its variable index vector.

Here is the encoding of the previous problem:

```
{
  "type": "moment",
  "variables": [
     "x",
```

```
    “y”
],
“nvar”: 2,
“npms”: 2,
"objective": {
    "set": "inf",
    "moments": {
          "nseq": 2,
          "coeftype": "Int64",
          "terms": [
              [1, 1 [4],[1]],
              [1, 1, [2,2]],
              [-1, 1, [3],[2]]
          ]
    }
},
"constraints": [
    {
          "set": ">=0",
          "moments": {
                "coeftype": "Float64",
                "terms": [
                      [1.0, 1, [2],[1]],
                      [3.141592653589793, 1, [2],[2]],
                      [-2.0, 1],
                      [2.0, 2, [2],[2]],
                      [-1.0, 2, [1],[2]]
                ]
          }
    },
    {
          "set": ">=0 *",
          "moments": {
                "coeftype": "Float64",
                "terms": [
                      [ 1.0, 1, [2],[1]],
                      [-1.0, 2,[1],[2]],
                      [-3.0, 0]
                ]
          }
    }
]
"version": "0.0.1",
"author": "Bernard Mourrain”,
```

```
  "uuid": "371ff738-9513-11ea-1f6b-9b491fe32502"
}
```

Here the problem is of type `"moment"` (Positive Moment Sequence) corresponding to the constraints $\mu_1 \succcurlyeq 0, \mu_2 \succcurlyeq 0$. Notice that a polynomial optimization problem can also be seen as a moment optimization problem with a single moment sequence μ_1 and a single mass constraint $\langle \mu_1, 1 \rangle = 1$.

SDP format

We consider the linear semidefinite optimization problem with p linear matrix inequalities and explicit linear equality constraints in the form

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} \ b^T x \quad \text{\#(SDP-P)} \\ &\text{subject to} \\ &\sum_{i=1}^{n} A_i^{(j)} x_i - A_0^{(j)} \succcurlyeq 0, \ \ j = 1, \ldots, p \\ &c_k^\top x - d_k \geq 0, \qquad k \in \mathcal{I} \\ &c_k^\top x - d_k = 0, \qquad k \in \mathcal{E} \end{aligned}$$

with data $b \in R^n$, $A_i^{(j)} \in \mathbb{S}^{m_j}$, $j = 1, \ldots, p$, $C \in \mathbb{R}^{q \times n}$, $d \in \mathbb{R}^q$. Here $\mathcal{I} \cap \mathcal{E} = \emptyset$, $|\mathcal{I}| + |\mathcal{E}| = q$, and the matrix C is formed by columns c_k, $k \in \mathcal{I} \cup \mathcal{E}$. Denote by $Y^{(j)}$ the dual variable associated with the j-th LMI, $Y^{(j)} \in \mathbb{S}^{m_j}$. The dual problem to (SDP-P) is then

$$\begin{aligned} &\max_{Y^{(1)} \ldots Y^{(p)}, z \in \mathbb{R}^q} \sum_{j=1}^{p} \left\langle A_0^{(j)}, Y^{(j)} \right\rangle + d^\top z \quad \text{\#(SDP-D)} \\ &\text{subject to} \\ &z_k \geq 0, \qquad k \in I. \end{aligned}$$

In addition to the data, the SDP input format may also contain the following information, specific to POP relaxations:

- $Y^{(j)}$ has (unknown) low rank for some specified $j = 1, \ldots, p$
- When known, Rank($Y^{(j)}$) for specified $j = 1, \ldots, p$
- $A_i^{(j)}$ is a low rank matrix defined by r vectors as $A_i^{(j)} = \sum_{k=1}^{r} (a_i^j)_k \, (a_i^j)_k^\top$

For SDP data, the `"type"` attribute is :

- `"type": "sdp"`

The `"objective"` part is the vector $[b_1, \ldots, b_p]$ representing the linear objective function.

The proposed SDP format to represent the constraints is based on the popular SDPA input format, where the data matrices are stored in a sparse format in an ASCII file. The `"constraints"` part has the following attributes:

- `"nlmi"`: number of linear matrix inequalities [p] (int)
- `"msizes"`: dimensions of the LMIs [$m_1, \ldots, m_p$]
- `"lmiduallr"`: (optional) binary vector of length `nlmi`, set to 1 if $Y^{(j)}$ assumed low rank, otherwise zero
- `"lmidualrank"`: (optional) vector of length `nlmi` with ranks of $Y^{(j)}$ (when known a-priori); if the rank of $Y^{(j)}$ is not known for some j, the value is zero
- `"lmi_symat"`: matrices $A_i^{(j)}$ in upper triangular sparse format encoded as a sequence of quintuples [$(A_i^{(j)})_{\square row,icol}, i, j, irow, icol$] if $(A_i^{(j)})_{\square row,icol} \neq 0$
- `"lmi_lrmat"`: matrices $A_i^{(j)}$ defined in low rank sparse format by r vectors $(a_i^j)_k$, $k = 1, \ldots, r$, such that $A_i^{(j)} = \sum_{k=1}^{r} (a_i^j)_k (a_i^j)_k^\top$ encoded as a sequence of quintuples [$((a_i^j)_k)_l, i, j, k, l$] if $((a_i^j)_k)_l \neq 0$
- `"nlsi"`: (optional) number of linear inequalities [q]
- `"lsi_mat"`: (optional) matrix C in sparse format encoded as a sequences of triples `[`[$(C)_{irow,icol}, irow, icol$] if $(C)_{\square row,icol} \neq 0$
- `"lsi_vec"`: (optional) vector d, [$d_1, \ldots, d_q$]
- `"lsi_op"`: (optional) binary vector of length `nlsi`, set to zero for equality constraints ($k \in \mathcal{E}$) and 1 for inequality ($k \in \mathcal{I}$) constraints; if this vector is not present and `nlsi>0`, inequality type constraints are assumed.

Example SDP1
Here is the encoding for the following SDP:

$$\min_{x \in \mathbb{R}^3} x_1 + 2\,x_2 + 3\,x_3$$

PMO format:

```
{
      "type": [
            "sdp"
      ],
      "nvar": 3,
      "objective": [1,2,3],
      "constraints": {
            "nlmi": 2,
            "msizes": [3,2],
            "lmi_symat": [
                  [ 1.0,0,2,1,2],
                  [ 2.0,0,2,2,2],
```

```
                [ 2.0,1,1,1,1],
                [ 2.0,1,1,2,2],
                [ 2.0,1,1,3,3],
                [-1.0,1,1,1,2],
                [ 1.0,1,2,1,1],
                [-1.0,1,2,2,2],
                [ 3.0,2,2,1,2],
                [ 2.0,3,1,1,1],
                [ 2.0,3,1,2,2],
                [ 2.0,3,1,3,3],
                [-1.0,3,1,1,3]
            ]
        },
        "version": "0.0.1",
        "uuid": "ad6dea66-95f8-11ea-0326-5727aaa79c1d",
        "name": [
            "My first example"
        ],
        "comment": [
            "Two LMIs"
        ]
}
```

Example SDP2

Here is the encoding for another SDP:

$$\min_{x\in\mathbb{R}^3} x_3$$
$$\text{subject to}$$
$$\begin{bmatrix}4 & 1 & 0\\ 1 & 1 & 0\\ 0 & 0 & 0\end{bmatrix} x_1 + \begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 1\\ 0 & 1 & 1\end{bmatrix} x_2 + \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 1\end{bmatrix} x_3 - \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{bmatrix} \succcurlyeq 0$$
$$x_1 + x_2 = 1, \qquad x_1 \geq 0, \qquad x_2 \geq 0$$

In addition, we know that the rank of the dual variable is equal to one.

PMO format:

```
{
        "type": [
            "sdp"
        ],
           "nvar": 3,
           "objective": [0,0,1],
```

```
         "constraints": {
           "nlmi": 1,
           "msizes": [3],
           "lmidualrank": 1,
           "lmi_symat": [
                [1.0,0,1,2,3],
                [4.0,1,1,1,1],
                [1.0,1,1,1,2],
                [1.0,1,1,2,2],
                [1.0,2,1,2,2],
                [1.0,2,1,2,3],
                [1.0,2,1,3,3]
           ],
           "lmi_lrmat": [
                [1.0,3,1,1,3]
           ],
           "nlsi": 3,
           "lsi_mat": [
                [1.0,1,1],
                [1.0,1,2],
                [1.0,2,1],
                [1.0,3,2]
           ],
           "lsi_vec": [1,0,0],
           "lsi_op [0,1,1]
         }
         "version": "0.0.1",
         "uuid": "9ee52456-9695-11ea-28bb-83c1d2b052cb",
         "name": "My second example",
         "comment": "One LMI with one rank-1 matrix, 3 linear
       scalar constraints, dual variable of rank one"
}
```


SDP format, relaxation specific

Several typical SDP problems arising from POP relaxations lead to the following simplified (SDP-D) problem

$$\max_{Y \in \mathbb{S}^m} \langle A_0, Y \rangle \quad \text{(SDP-D-rel)}$$
$$\text{subject to}$$
$$\langle A_i, Y \rangle \leq b_i, \qquad i \in \mathcal{I}$$

$$\langle A_i, Y\rangle = b_i, \qquad i \in \mathcal{E}$$
$$Y \succcurlyeq 0$$

with $\mathcal{I} \cap \mathcal{E} = \emptyset,\ |\mathcal{I}| + |\mathcal{E}| = n.$ In terms of (SDP-P), this means that we have only one LMI and zero lower bounds on some of the variables, those with $i \in \mathcal{I}$.

For convenience, we use a specific SDP input format tailored to (SDP-D-rel). A (SDP-D-rel) problem is represented by the following attributes.

The `"type"` attribute is :

- `"type": "sdp_relax"`

The `"objective"` attribute is a component with the following attributes.

- `"msizes"`: dimension of the matrix variable
- `"symat"`: matrix A_0 in upper triangular sparse format encoded as a sequence of trituples $[(A_0)_{irow,icol}, irow, icol]$ if $(A_0)_{irow,icol} \neq 0$

The `"constraints"` attribute is a component with the following attributes.

- `"ncon"`: total number of constraints [n]
- `"lmilr"`: (optional) binary, set to 1 if Y assumed low rank, otherwise zero
- `"lmirank"`: (optional) rank of Y (when known a-priori)
- `"rhs"`: right-hand side vector [$b_1, \ldots, b_n$]
- `"symat"`: matrices $A_i^{(j)}$ in upper triangular sparse format encoded as a sequence of quintuples $[(A_i^{(j)})_{irow,icol}, i, 1, irow, icol]$ if $(A_i^{(j)})_{irow,icol} \neq 0$
- `"lrmat"`: matrices $A_i^{(j)}$ defined in low rank sparse format by r vectors $(a_i^j)_k,\ k = 1, \ldots, r,$ such that $A_i^{(j)} = \sum_{k=1}^{r} (a_i^j)_k (a_i^j)_k^\top$ encoded as a sequence of quintuples $[((a_i^j)_k)_l, i, 1, k, l]$ if $((a_i^j)_k)_l \neq 0$
- `"op"`: binary vector of length `ncon`, set to zero for equality constraints ($i \in \mathcal{E}$) and 1 for inequality ($i \in \mathcal{I}$) constraints; if this vector is not present, inequality type constraints are assumed.

Example SDP3
Consider the following SDP:

$$\max_{Y \in \mathbb{S}^3} \left\langle \begin{bmatrix} 4 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & 4 \end{bmatrix}, Y \right\rangle$$
subject to
$$\langle A_1, Y\rangle \leq 1$$
$$\langle A_2, Y\rangle = 1$$
$$Y \succcurlyeq 0$$

with matrices $A_i = a_i a_i^T$ and

$$a_1 = (2\ 1\ 1)^T \quad a_2 = (3\ 2\ 1)^T.$$

PMO format:

```
{
            "type": [
               "sdp_relax"
            ],
            "objective":{
                "msizes": 3,
                "symat": [
                    [ 4.0,0,1,1,1],
                    [ 1.0,0,1,1,2],
                    [-2.0,0,1,2,2],
                    [ 1.0,0,1,2,3],
                    [ 4.0,0,1,3,3]
                ]
            }
            "constraints":{
                "ncon": 2,
                "rhs": [1.0 1.0],
                "lrmat": [
                    [2.0,1,1,1,1],
                    [1.0,1,1,1,2],
                    [1.0,1,1,1,3],
                    [3.0,2,1,1,1],
                    [2.0,2,1,1,2],
                    [1.0,2,1,1,3]
                ],
                "op": [1,0]
          },
          "name": [
              "My third example"
          ],
}
```

Matlab package

Matlab Poema ASCII format

A problem can either be encoded using `json` format or in Matlab, as a Matlab structure with fields equal to the `json` attributes. The two formats can be converted from/to each other using the Matlab toolbox “jsonlab” https://uk.mathworks.com/matlabcentral/fileexchange/33381-jsonlab-a-toolbox-to-encode-decode-json-files .
Furthermore, convertors from/to the SDPA format are provided, so that the problems can also be solved using existing software such as MOSEK or SeDuMi.

Assume that the example SDP1 (see section SDP format) is stored in a file `myfile.json`. To read it as a Matlab `char` array named `sdp_json` and to convert it into a Matlab structure, call

```
>> sdp_json = fileread('myfile.json');
>> my_sdp_structure = loadjson(sdp_json);
```

Below is the result, problem SDP1 in Matlab format.

Matlab format

```
sdp1 = struct(...
    "type","sdp",...
    "nvar", [3],...
    "objective", [ 1.0 2.0 3.0 ],...
    "constraints",struct(...
        "nlmi", [2],...
        "msizes", [ 3 2 ],...
        "lmi_symat", [...
        [ 1.0 0 2 1 2] ;...
        [ 2.0 0 2 2 2] ;...
        [ 2.0 1 1 1 1] ;...
        [ 2.0 1 1 2 2] ;...
        [ 2.0 1 1 3 3] ;...
        [-1.0 1 1 1 2] ;...
        [ 1.0 1 2 1 1] ;...
        [-1.0 1 2 2 2] ;...
        [ 3.0 2 2 1 2] ;...
        [ 2.0 3 1 1 1] ;...
        [ 2.0 3 1 2 2] ;...
        [ 2.0 3 1 3 3] ;...
        [-1.0 3 1 1 3] ;...
    ]),...
    "name","My first example",...
    "comment","Two LMIs"...
    )
```

To convert the above structure back into `json` format and write it into an ASCII file, call

```
>> sdp_json = savejson('sdp',sdp1);  %the output is a character array
>> fileID = fopen('myfile.txt','w');
>> fprintf(fileID,'%s\n',sdp_json)
```

Matlab format for SDP2 is as follows:

```
sdp2 = struct(...
    "type","sdp",...
    "nvar", [3],...
    "objective", [ 0.0 0.0 1.0 ],...
    "constraints",struct(...
    "nlmi", [1],...
    "msizes", [ 3 ],...
    "lmidualrank", [1],...
    "lmi_symat", [...
    [1.0 0 1 2 3] ;...
    [4.0 1 1 1 1] ;...
    [1.0 1 1 1 2] ;...
    [1.0 1 1 2 2] ;...
    [1.0 2 1 2 2] ;...
    [1.0 2 1 2 3] ;...
    [1.0 2 1 3 3] ;...
    ],...
    "lmi_symat", [...
    [1.0 3 1 1 3] ;...
    ],...
    "nlsi", [3],...
    "lsi_mat", [...
    [1.0 1 1] ;...
    [1.0 1 2] ;...
    [1.0 2 1] ;...
    [1.0 3 2] ;...
    ],...
    "lsi_rhs", [ 1.0 0.0 0.0 ],...
    "lsi_op", [ 0 1 1] ...
    ),...
    "name","My second example",...
    "comment","One LMI with one rank-1 matrix, 3 linear scalar
constraints, dual variable of rank one"...
    );
```

Matlab format for SDP3 is as follows:

```
sdp = struct(...
```

```
    "type","sdp_relax",...
    "objective",struct(...
        "msizes", [3],...
        "symat", [
        [ 4.0 0 1 1 1];...
        [ 1.0 0 1 1 2];...
        [-2.0 0 1 2 2],
        [ 1.0 0 1 2 3];...
        [ 4.0 0 1 3 3];...
        ]...
    ),...
    "constraints",struct(...
        "ncon", [2],...
        "rhs", [1 1],...
        "lrmat", [...
        [2.0 1 1 1 1] ;...
        [1.0 1 1 1 2] ;...
        [1.0 1 1 1 3] ;...
        [3.0 2 1 1 1] ;...
        [2.0 2 1 1 2] ;...
        [1.0 2 1 1 3] ;...
        ],...
        "op", [1 0] ...
    ),...
    "name","My third example"...
    );
```

Matlab sparse format

The second Matlab format is based on Matlab sparse arrays and is to be used as input for POEMA SDP software coded in Matlab. All data are again stored in a Matlab structure but while most of the attributes remain the same, sparse matrices and vectors are converted to sparse Matlab arrays.

Problem SDP1 in Matlab sparse format is stored in the following structure:

Matlab sparse format

```
struct with fields:
     type: "sdp"
     nvar: 3
     name: "My first example"
  comment: "Two LMIs"
        c: [1 2 3]
     nlmi: 2
   msizes: [3 2]
```

```
          A: {2×4 cell}
```

where cells in field `A` contain data matrices $A_i^{(j)},\ i = 1, \ldots n,\ j = 1, \ldots, p$, stored in sparse Matlab format.

Matlab sparse format for problem SDP2

```
  struct with fields:
       type: "sdp"
       nvar: 3
       name: "My second example"
    comment: "One LMI with one rank-1 matrix, 3 linear scalar
constraints, dual variable of rank one"
          c: [0 0 1]
       nlmi: 1
     msizes: 3
       nlin: 3
     lsi_op: [0 1 1]
          A: {[3×3 double]  [3×3 double]  [3×3 double]  [3×3
double]}
       Avec: {[3×1 double]  [3×1 double]  [3×1 double]  [3×1
double]}
          C: [3×2 double]
          d: [1 0 0]
```

Matlab sparse format for problem SDP3

```
  struct with fields:
       type: "sdp_relax"
       name: "My third example"
    comment: "Dual SDP with one inequality"
          c: [1 1]
     msizes: 3
       nlmi: 1
       nvar: 2
          A: {[3×3 double]  [3×3 double]  [3×3 double]}
       Avec: {[3×1 double]  [3×3 double]  [3×3 double]}
       nlin: 1
          C: [1 0]
          d: 0
```

We provide the following Matlab conversion functions:

```
function sdp = sdpa2poema(filename);
% Reads linear SDP problem from an SDPA file, separates the linear
% constraint matrix, returns the problem in the POEMA Matlab structure

function sdp = poema2sdpa(filename);
% Input: SDP problem in the POEMA Matlab structure
% Output: ASCII file with problem in SDPA format
% Any information special to POEMA format (low rank) is ignored

function sdpdata = poema2sparse(poema_struct);
% Input: SDP problem in the POEMA Matlab structure
% Output: Matlab structure with data matrices stored as Matlab sparse
% matrices
```

Registries

The data are registered in the file registries/index-pmo.csv of the public repository https://github.com/PolynomialMomentOptimization/data.
They are stored as a tuple of

- An Universally Unique IDentifier or UUID, which is a string uniquely identifying the data resource.
- A name, which is a string (that can be "")
- A data URL where the resource file can be recovered.
- A documentation URL (optional) where information on the data can be found. It can be a human readable document, like a documentation, an article about the data.

The table of association between UUID and Data is stored Resources in the file registries/index-pmo.csv in the CSV format (Column Separated Values). The first column is the UUID, the second column is the name, the third is the URL of the data resource and the fourth column is the URL of the metadata information (Optional). The metadata can also be stored in a json format and give information about the date, the author and other features of the data that we want to exhibit.

Database access and management

The data files and code for creating and managing data are publically available on the server https://github.com/PolynomialMomentOptimization/.
Every member of the network shall have an account on the server to create and manage the data there.

New data can be added by any one, using the mechanism of pull request on the github server. The pull request will then be examined and validated or rejected depending on the validity of request.

Tables with properties of the PMO problems can also be built and accessed with a similar mechanism.

License

A public copyright license such as Creative Commons, will be attached to the data stored in the repositories, to allow the public access and the free distribution of the documents, which are the files storing the data. A typical example can be *Creative Commons BY-NC-SA* with the author information, Non-Commercial use and SharedAlike :  .

Code for the database

Code for writing, reading data in the JSON format and accessing the database are developed for Julia and Matlab

Julia Package

A julia package `PMO.jl` for registering new data and using the PMO data is available at `https://github.com/PolynomialMomentOptimization/POP.jl`.
See https://github.com/PolynomialMomentOptimization/PMO.jl/expl/Tutorial.ipynb.

Matlab package

A dedicated matlab package is available at `https://github.com/PolynomialMomentOptimization/Matlab-suite`. Here are some illustrations of its use.

Matlab Poema ASCII format

A problem can either be encoded using `json` format or in Matlab, as a Matlab structure with fields equal to the `json` attributes. The two formats can be converted from/to each other using the Matlab toolbox “jsonlab” https://uk.mathworks.com/matlabcentral/fileexchange/33381-jsonlab-a-toolbox-to-encode-decode-json-files .

Furthermore, convertors from/to the SDPA format are provided, so that the problems can also be solved using existing software such as MOSEK or SeDuMi.

Assume that the example SDP1 (see section SDP format) is stored in a file `myfile.json`. To read it as a Matlab `char` array named `sdp_json` and to convert it into a Matlab structure, call

```
>> sdp_json = fileread('myfile.json');
>> my_sdp_structure = loadjson(sdp_json);
```

Below is the result, problem SDP1 in Matlab format.

Matlab format

```
sdp1 = struct(...
    "type","sdp",...
    "nvar", [3],...
    "objective", [ 1.0 2.0 3.0 ],...
    "constraints",struct(...
        "nlmi", [2],...
        "msizes", [ 3 2 ],...
        "lmi_symat", [...
        [ 1.0 0 2 1 2] ;...
        [ 2.0 0 2 2 2] ;...
        [ 2.0 1 1 1 1] ;...
        [ 2.0 1 1 2 2] ;...
        [ 2.0 1 1 3 3] ;...
        [-1.0 1 1 1 2] ;...
        [ 1.0 1 2 1 1] ;...
        [-1.0 1 2 2 2] ;...
        [ 3.0 2 2 1 2] ;...
        [ 2.0 3 1 1 1] ;...
        [ 2.0 3 1 2 2] ;...
        [ 2.0 3 1 3 3] ;...
        [-1.0 3 1 1 3] ;...
    ]),...
    "name","My first example",...
    "comment","Two LMIs"...
    )
```

To convert the above structure back into `json` format and write it into an ASCII file, call

```
>> sdp_json = savejson('sdp',sdp1);  %the output is a character array
>> fileID = fopen('myfile.txt','w');
>> fprintf(fileID,'%s\n',sdp_json)
```

Matlab format for SDP2 is as follows:

```
sdp2 = struct(...
    "type","sdp",...
    "nvar", [3],...
    "objective", [ 0.0 0.0 1.0 ],...
    "constraints",struct(...
    "nlmi", [1],...
    "msizes", [ 3 ],...
    "lmidualrank", [1],...
    "lmi_symat", [...
    [1.0 0 1 2 3] ;...
    [4.0 1 1 1 1] ;...
    [1.0 1 1 1 2] ;...
    [1.0 1 1 2 2] ;...
    [1.0 2 1 2 2] ;...
    [1.0 2 1 2 3] ;...
    [1.0 2 1 3 3] ;...
    ],...
    "lmi_symat", [...
    [1.0 3 1 1 3] ;...
    ],...
    "nlsi", [3],...
    "lsi_mat", [...
    [1.0 1 1] ;...
    [1.0 1 2] ;...
    [1.0 2 1] ;...
    [1.0 3 2] ;...
    ],...
    "lsi_rhs", [ 1.0 0.0 0.0 ],...
    "lsi_op", [ 0 1 1] ...
    ),...
    "name","My second example",...
    "comment","One LMI with one rank-1 matrix, 3 linear scalar
constraints, dual variable of rank one"...
    );
```

Matlab format for SDP3 is as follows:

```
sdp = struct(...
    "type","sdp_relax",...
    "objective",struct(...
        "msizes", [3],...
        "symat", [
        [ 4.0 0 1 1 1];...
        [ 1.0 0 1 1 2];...
        [-2.0 0 1 2 2],
        [ 1.0 0 1 2 3];...
        [ 4.0 0 1 3 3];...
        ]...
    ),...
```

```
  "constraints",struct(...
      "ncon", [2],...
      "rhs", [1 1],...
      "lrmat", [...
      [2.0 1 1 1 1] ;...
      [1.0 1 1 1 2] ;...
      [1.0 1 1 1 3] ;...
      [3.0 2 1 1 1] ;...
      [2.0 2 1 1 2] ;...
      [1.0 2 1 1 3] ;...
      ],...
      "op", [1 0] ...
  ),...
  "name","My third example"...
  );
```

Matlab sparse format

The second Matlab format is based on Matlab sparse arrays and is to be used as input for POEMA SDP software coded in Matlab. All data are again stored in a Matlab structure but while most of the attributes remain the same, sparse matrices and vectors are converted to sparse Matlab arrays.

Problem SDP1 in Matlab sparse format is stored in the following structure:

Matlab sparse format

```
struct with fields:
     type: "sdp"
     nvar: 3
     name: "My first example"
  comment: "Two LMIs"
        c: [1 2 3]
     nlmi: 2
   msizes: [3 2]
        A: {2×4 cell}
```

where cells in field `A` contain data matrices $A_i^{(j)},\ i = 1, \ldots n,\ j = 1, \ldots, p$, stored in sparse Matlab format.

Matlab sparse format for problem SDP2

```
  struct with fields:
       type: "sdp"
       nvar: 3
       name: "My second example"
    comment: "One LMI with one rank-1 matrix, 3 linear scalar
constraints, dual variable of rank one"
          c: [0 0 1]
       nlmi: 1
     msizes: 3
       nlin: 3
     lsi_op: [0 1 1]
          A: {[3×3 double]  [3×3 double]  [3×3 double]  [3×3
double]}
       Avec: {[3×1 double]  [3×1 double]  [3×1 double]  [3×1
double]}
          C: [3×2 double]
          d: [1 0 0]
```

Matlab sparse format for problem SDP3

```
  struct with fields:
       type: "sdp_relax"
       name: "My third example"
    comment: "Dual SDP with one inequality"
          c: [1 1]
     msizes: 3
       nlmi: 1
       nvar: 2
          A: {[3×3 double]  [3×3 double]  [3×3 double]}
       Avec: {[3×1 double]  [3×3 double]  [3×3 double]}
       nlin: 1
          C: [1 0]
          d: 0
```

We provide the following Matlab conversion functions:

```
function sdp = sdpa2poema(filename);
% Reads linear SDP problem from an SDPA file, separates the linear
% constraint matrix, returns the problem in the POEMA Matlab structure

function sdp = poema2sdpa(filename);
% Input: SDP problem in the POEMA Matlab structure
```

```
% Output: ASCII file with problem in SDPA format
% Any information special to POEMA format (low rank) is ignored

function sdpdata = poema2sparse(poema_struct);
% Input: SDP problem in the POEMA Matlab structure
% Output: Matlab structure with data matrices stored as Matlab sparse
% matrices
```