SPOT
(Systems Polynomial Optimization Tools)
Manual

Alexandre Megretski †*

January 11, 2010

SPOT (Systems Polynomial Optimization Tools) is a MATLAB toolbox written as an alternative implementation of SOSTools to be used in implementing a class of nonlinear system identification algorithms. It was tested with `MATLAB 7.8.0` (R2009a). SPOT provides its own matrix multivariable polynomial variable class `msspoly` for handling elementary polynomial operations, a special class `mssprog` for defining convex optimization problems (to be solved by `SeDuMi`) in terms of polynomial identities and self-dual cones, and a set of functions for identification of linear and nonlinear dynamical systems.

1 Installation

POT is distributed in the form of compressed archives `spotDDMMYY.zip`, where `DDMMYY` indicates the date of release (for example, `pot110110.zip` was released on January 11, 2010).

Create directory `spot` and extract `spotDDMMYY.zip` into it. Start MATLAB, and run `spot_install.m` from the `pot` directory. The script sets the path for POT, and compiles some binaries:

```
>> spot_install

 Installing SPOT in C:\home\matlab\spot:
```

* †LIDS, EECS, MIT, ameg@mit.edu

```
 updating the path...
 compiling the binaries...
 Done.
>>
```

Once SPOT is installed, you can check that it works by running `spot_chk.m`:

```
>> spot_chk
```

2 Multivariable Polynomials

The `@msspoly` environment handles matrix polynomials in multiple variables. Individual variables in `@msspoly` have identifiers which begin with a *single* character, which may be followed by a non-negative integer number. While, in principle, a variable identifier can begin with any MATLAB-recognized character, the only ones which are safe to use in applications are the 52 Latin alphabet letters

`ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz`

Other characters can be used for automated variable definitions. For example, in the `mssprog` environment (used to define convex optimization programs in terms of polynomial equations which are linear with respect to the optimized (*decision*) variables, but have arbitrary dependence on other (*abstract*) variables, the identifiers beginning with a "`@`" are reserved for hidden decision variables, while the identifiers beginning with a "`#`" are reserved for hidden abstract variables.

2.1 Defining `@msspoly` Variables

Use `msspoly.m`:

```
>> f=msspoly('v')

[    v  ]
```

defines `xx` as an `@msspoly` polynomial `f=` $f : f(v) = v$. `g=msspoly('v',k)` where `k` is a positive integer will define a `k`-by-1 vector of different variables, as in

```
>> g=msspoly('t',3)

[   t0  ]
[   t1  ]
[   t2  ]
```

Using `g=msspoly('v',[a b])` prodices a vector of `a` variables with indexing starting with `b`, as in

```
>> g=msspoly('A',[2 1])

[   A1  ]
[   A2  ]
```

2.2 "Free" and "Simple" `@msspoly` Variables

An `@msspoly` variable is called *free* if it is a matrix of independent scalar variables. An `@msspoly` variable is called *simple* if it is a column of independent scalar variables and constants. For example, in

```
>> f1=msspoly('x',7);
>> f2=f1(1:6);
>> f3=reshape(f1,3,2);
>> f4=[f2;f2;1];
>> f5=f1*f1';
```

the resulting free variables are `f1`, `f2`, `f3`, the simple variables are `f1`, `f2`, `f3`, `f4`, while `f5` is not free and not simple.

2.3 Handling `@msspoly` Variables

A number of functions of the `@msspoly` environment have the standard meaning:

- `ctranspose.m` (as in `z=x'`)
- `horzcat.m` (as in `z=[x y]`)
- `minus.m` (as in `z=x-y`)
- `mtimes.m` (as in `z=x*y`)

- `plus.m` (as in `z=x+y`)
- `uminus.m` (as in `z=-x`)
- `uplus.m` (as in `z=+x`)
- `vertcat.m` (as in `z=[x;y]`)
- `subsasgn.m` (as in `x(2)=y`)
- `reshape.m`
- `isempty.m`
- `isscalar.m`
- `length.m`
- `repmat.m`
- `size.m`
- `sum.m`

Other functions are close to their expected definitions, with minor modifications or restrictions:

- `decomp.m`: decomposes an `@msspoly` variable into a vector of its free variables, and matrices of degrees and coefficients of its terms;
- `deg.m`: gives the a single number degree; can be used with a second argument, which must be a *free* `@msspoly` variable, in which case the degree with respect to the independent variables listed in the second argument is computed;
- `diag.m`: produces a diagonal `@msspoly` matrix when the input is a row or a column; otherwise extracts the diagonal as a column vector;
- `diff.m`: the second (required) and third (optional) arguments must be free `@msspoly` variables; with two arguments, the first argument must be a column, and the result is the matrix of the partial derivatives of the first argument with respect to the second; with three arguments, the first argument can have arbitrary dimensions, and the result is the derivative of the first argument with respect to the second in the direction provided by the third;

- `double.m`: converts a constant `@msspoly` to `double`, otherwise returns character '?';
- `isfunction.m`: the second argument must be a free `@msspoly`; true iff the firts argument is a function of the second;
- `mono.m`: produces column vector of all monomials from the argument;
- `mpower.m` (as in `z=x^y`): argument `y` must be a non-negative integer;
- `mrdivide.m` (as in `z=x/y`): argument `y` must be a non-singular `double`;
- `newton.m`: applies Newton method iterations to try to solve approximately systems of polynomial equations;
- `recomp.m`: the inverse of `decomp.m`;
- `subs.m`: a restricted substitution routine, allows to replace, in the first argument, the independent variables from the second argument (must be a *free* `@msspoly`) by the corresponding components of the third argument (must be a *simple* `@msspoly`);
- `subsref.m` (as in `z=x(1:2)` or `z=x.n`): for the first type of call, works as expected; for the second, `x.m` and `x.n` return the dimensions, while `x.s` returns the internal `@msspoly` structure of `x` (something that only a developer of new `@msspoly` code would need);
- `trace.m`: the usual sum of the diagonal elements, but non-square arguments are admissible, too.

3 MSS Programs

MSS stands for "Modified Sums of Squares"[1] The `@mssprog` environment allows its user to define matrix decision variables ranging over certain convex sets which are, in `SeDuMi` terminology, self-dual cones, to impose linear constraints in terms of polynomial identities, to call `SeDuMi` to optimize the decision variables, and to extract the resulting optimal values.

[1]or for "Meager Sums of Squares", "Magnificent Sums of Squares", etc., just not "Alternative Sums of Squares", though that's what it is.

3.1 `@mssprog` Operations

To initialize a blanc MSS program, use `mssprog.m`, as in

```
pr=mssprog;
```

The most straightforward way of adding items to an MSS program is by using the `'.'` subsassignments:

- `pr.free=x` registers the elements of `x` as free (`SeDuMi`-style) decision variables (`x` must be a *free* `@msspoly`);
- `pr.pos=x` registers the elements of `x` as positive (`SeDuMi`-style) decision variables (`x` must be a *free* `@msspoly`);
- `pr.lor=x` registers the columns of `x` as Lorentz cone (`SeDuMi`-style) decision variables (`x` must be a *free* `@msspoly` with at least two rows);
- `pr.rlor=x` registers the columns of `x` as rotated Lorentz cone (`SeDuMi`-style) decision variables (`x` must be a *free* `@msspoly` with at least three rows);
- `pr.psd=x` registers the elements of every column of `x` as the components of positive semidefinite (`SeDuMi`-style) decision variables (`x` must be a *free* `@msspoly` with `nchoosek(m+1,2)` rows to generate an `m`-by-`m` symmetric matrix: use `y=mss_v2s(x(:,k)` to re-shape the `k`-th column of `x` into the corresponding symmetric matrix);
- `pr.eq=x` registers equality `x==0` with MSS program `pr` (`x` must be an `@msspoly` which is linear with respect to the vector of all independent variables which are registered with `pr` as decision parameters);
- `pr.sos=x` registers the constraint that all scalar components of x must be sums of squares of polynomials (`x` must be an `@msspoly` which is linear with respect to the vector of all independent variables which are registered with `pr` as decision parameters);
- `pr.sss=x` registers the constraint that all `u'*x*u`, where `u=msspoly('#',size(x,1))`, must be a sum of squares (`x` must be a square-sized `@msspoly` which is linear with respect to the vector of all independent variables which are registered with `pr` as decision parameters);

- `pr.sedumi=r` calls `SeDuMi` to find the values of the decision variables which minimize `r` (`r` must be a scalar `@msspoly` which is a linear function of the decision parameters).

To extract the optimized polynomials, use the `'()'` subsreferencing:

- `y=pr(x)`: `y` is the result of substituting the optimized values of decision variables into `@msspoly` x;
- `y=pr({x})`: same as `double(pr(x))`.

For example, the following code (contained in `mss_test3.m`) finds the minimal value of r for which the polynomial $4x^4y^6 + rx^2 - xy^2 + y^2$ is a sum of squares:

```
x=msspoly('x');                          % define the variables
y=msspoly('y');
r=msspoly('r');
q=4*(x^4)*(y^6)+r*(x^2)-x*(y^2)+y^2; % the SOS polynomial
pr=mssprog;                              % initialize MSS program
pr.free=r;                               % register r as free
pr.sos=q;                                % register sos constraint
pr.sedumi=r;                             % minimize r
pr({r})                                  % get the optimal r
```

4 System Identification

Functions from the `nlid` directory are designed to help solving nonlinear system identification problems. Functions from the `nlid` directory treat the linear time invariant (LTI) case.

4.1 Identification of Symmetric Passive Transfer Matrices

Function `ltid_passive.m` is a user interface to a number of algorithms to aid in converting frequency samples of a marginally stable symmetric transfer matrix (such as impedance of a passive circuit) to a reduced order state space model.

4.2 Nonlinear System Identification

Currently, the following examples from the `nlid` directory appear to work:

- `nlid_io_lti_old_test1.m`: LTI system identification with POT;
- `nlid_miso0_test1.m`: memoryless NL system identification;
- `nlid_fl_test1.m`: more powerful memoryless NL system identification.