# POCP: a package for
# Polynomial Optimal Control Problems

Didier Henrion, Jean B. Lasserre and Carlo Savorgnan *

User's guide. Version 1.0 of July 1, 2008

**Abstract**

POCP is a new Matlab package running jointly with GloptiPoly 3 and, option-
ally, YALMIP. It is aimed at nonlinear optimal control problems for which all the
problem data are polynomial, and provides an approximation of the optimal value
as well as some control policy. Thanks to a user-friendly interface, POCP reformu-
lates such control problems as generalized problems of moments, in turn converted
by GloptiPoly 3 into a hierarchy of semidefinite programming problems whose asso-
ciated sequence of optimal values converges to the optimal value of the polynomial
optimal control problem. In this paper we describe the basic features of POCP and
illustrate them with some numerical examples.

# 1 What is POCP?

POCP is a Matlab package aimed at solving approximately nonlinear optimal control
problems (OCPs) for which all the problem data are polynomial. Consider a continuous-
time system described by the differential equation

$$\dot{x}(t) = f(t, x(t), u(t)), \tag{1}$$

1

where $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ are the state and input vectors, respectively. Given the running cost $h : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ and the final cost $H : \mathbb{R}^n \to \mathbb{R}$, the total cost to be minimized in the OCP is defined as

$$J(0, T, x(0), u(\cdot)) = \int_0^T h(t, x(t), u(t))dt + H(x(T)), \tag{2}$$

where $T$ is the horizon length. Different OCPs can be formulated depending if the initial condition $x(0)$ has been assigned or not.

## 1.1 Optimal control problem with free initial condition

Consider the following constraints:

- $x(0) \in \mathcal{C}_I = \{x : g_{I_j}(x) \leq 0, \ j = 1, \ldots, n_I\}$;

- $x(T) \in \mathcal{C}_F = \{x : g_{F_j}(x) \leq 0, \ j = 1, \ldots, n_F\}$;

- $(t, x(t), u(t)) \in \mathcal{C}_T = \{(t, x, u) : g_{T_j}(t, x, u) \leq 0, \ j = 1, \ldots, n_T\}$.

An OCP with free initial condition is of the form

$$
\begin{aligned}
&\min_{x(0), u(t)} \quad J(0, T, x(0), u(t)) \\
&\qquad x(0) \in \mathcal{C}_I \\
&\qquad (t, x(t), u(t)) \in \mathcal{C}_T \\
&\qquad x(T) \in \mathcal{C}_F
\end{aligned}
\tag{3}
$$

To solve this problem using POCP all the data must be polynomial. More specifically, $f$, $h$, $H$, $g_{I_j}$, $g_{T_j}$, and $g_{F_j}$ must be polynomial functions.

## 1.2 Optimal control problem with fixed initial condition

When set $\mathcal{C}_I$ contains only one point, the initial condition is fixed. From this viewpoint, the minimization problem (3) can be used also to formulate OCPs where the initial condition is fixed.

However, it can be interesting to consider a more general framework where the initial condition is not known exactly, but stochastically. In this case, the probability measure of the initial condition $\mu_I$ is given, and we consider the following OCP

$$
\begin{aligned}
&\min_{u(\cdot)} \quad \int J(0, T, x_0, u(x_0, t))d\mu_I(x_0) \\
&\qquad (t, x(t), u(t)) \in \mathcal{C}_T \\
&\qquad x(T) \in \mathcal{C}_F.
\end{aligned}
\tag{4}
$$

To better understand how to use this formulation, see [1].

POCP can also deal with problems where some state variables are fixed at time $t = 0$ and some of them can be chosen inside a set $\mathcal{C}_I$.

POCP can be used for problems where the horizon $T$ is fixed or not.

# 2  How does POCP work and what does it calculate?

A full explanation of the theory on which POCP is based is out of scope for this user's guide. The interested reader is referred to [1] and [2].

The techniques implemented in POCP are based on a modeling via occupation measures associated with semidefinite programming (SDP) relaxations. POCP formulates a hierarchy of relaxations giving a sequence of lower bounds on the optimal value of the OCPs (3) and (4). Furthermore, POCP returns the moment matrices of the occupation measures used and, as a byproduct, a polynomial subsolution of the Hamilton-Jacobi-Bellman (HJB) equation.

# 3  Installation

POCP is a free Matlab package consisting of an archive file downloadable from

`www.laas.fr/~henrion/software/pocp`

The installation consists of two steps:

- extract the directory `@pocp` from the archive file,

- copy `@pocp` on your working directory or, using the command `addpath`, add the parent directory of `@pocp` to the Matlab path.

POCP is based on GloptiPoly version 3.3 or higher [3]. Therefore it is assumed that this package is properly installed. To compute polynomial subsolutions of the HJB equation, the optimization modelling toolbox YALMIP [4] must be installed as well.

# 4 An introductory example

Consider the double integrator

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

with state constraint $x_2(t) \geq -1$ and input constraint $-1 \leq u(t) \leq 1$. When we consider the problem of steering in minimum time the state to the origin, an analytic solution to this problem is available [2]. When $x(0) = [1\ 1]^T$, the value of the minimum time to reach the origin is $T = 3.5$. In the following, we show a Matlab POCP script which defines the problem and calculates a lower bound on the optimal value.

```
% system variables
mpol x 2;  % state vector of size 2
mpol u; % scalar input

% state and input matrices
A = [0 1; 0 0];
B = [0; 1];

% define optimal control problem
prob = pocp( ...
        'state', x, ...
        'input', u, ...
        'dynamics', A*x+B*u);

% set further properties
prob = set(prob, 'idirac', x, [1; 1], ... % initial condition
                 'fdirac', x, [0; 0], ... % final condition
                 'tconstraint', [x(2)>=-1; u>=-1; u<=1], ... % constraints
                 'scost', 1);  % setting integral cost h to 1

% call to the solver
[status, cost] = solvepocp(prob, 14); % 14 = degree of moment matrix

disp('The lower bound on the minimum time is');
disp(cost);
```

We can notice that the user is required to know only four commands: `mpol` (a GloptiPoly command to define variables), `pocp`, `set` and `solvepocp`. When the script file is executed, we obtain the following output

...

```
The lower bound on the minimum time is
    3.4988
```

In the following section we illustrate all POCP commands together with examples of their usage.

# 5    Command guide

POCP consists of three commands: `pocp`, `set`, and `solvepocp`.

## 5.1    The `pocp` command

**Short description:** Creates a new POCP object.
**Syntax and usage:** When used in the form

```
>> prob = pocp;
```

the command `pocp` creates an empty POCP object `prob` of class `pocp`. The problem data can be then specified using the command `set`.

The `pocp` command can also be used to create a POCP object and, at the same time, set the problem data. In this case, the syntax is

```
>> prob = pocp('Property1', Argument1, Argument2, ...,'Property2', ...)
```

The properties and the arguments should be specified accordingly to the syntax of the `set` command, see below.

## 5.2    The `set` command

**Short description:** Sets the data of a POCP.
**Syntax and usage:** Specifies one or more properties of the POCP object. For example, to set a property which requires two arguments, use

```
>> prob = set(prob, 'Property1', Argument1, Argument2);
```

To set more properties at once,

```
>> prob = set(prob, 'Property1', Argument1, ..., 'Property2', ...);
```

To help the user remember the name of the properties the following convention has been used: "i" refers to the initial condition, "f" to the final condition, "t" to the trajectory, and "s" (for sum) to an integral.

In the sequel we list and describe all the properties that can be specified and their arguments.

### 5.2.1 Setting the state variables

**Property name:** state.
**Number of arguments:** 1.
**Default value:** empty.
**Description:** Specifies state variables, a vector of GloptiPoly class mpol.
**Example**
In this example we declare a variable x of dimension 2 and, after creating a POCP object prob, we set x as the state variable.

```
>> mpol x 2;
>> prob = pocp;
>> prob = set(prob, 'state', x);
```

State variables can also have different names as in the following example

```
>> mpol x1 x2;
>> prob = pocp;
>> prob = set(prob, 'state', [x1; x2]);
```

For more information on declaring GloptiPoly variables type

```
>> help mpol
```

or check [3].

### 5.2.2  Setting the input variables

**Property name:** `input`.
**Number of arguments:** 1.
**Default:** empty.
**Description:** Specifies input variables, a vector of GloptiPoly class `mpol`.
**Example**
Suppose a POCP object `prob` has been already defined. With the following commands a scalar `mpol` variable `u` is defined and then set as the input variable.

```
>> mpol u;
>> prob = set(prob, 'input', u);
```

### 5.2.3  Setting the time variable

**Property name:** `time`.
**Number of arguments:** 1.
**Default:** empty.
**Description:** Specifies the time variable, a scalar of GloptiPoly class `mpol`. When the time variable is not specified but is needed internally by POCP, then it is defined automatically.
**Example**
Suppose a POCP object `prob` has already been defined. With the following commands a scalar `mpol` variable `t` is defined and then set as the time variable.

```
>> mpol t;
>> prob = set(prob, 'time', t);
```

### 5.2.4  Setting the dynamics

**Property name:** `dynamics`.
**Number of arguments:** 1.
**Default:** empty.
**Description:** Specifies the system dynamics, a vector of GloptiPoly class `mpol` of the same size as the state vector.
**Example**
Assume a POCP object `prob` has been created and the state vector `x` and input variable `u` have been set. In this example we set the linear dynamics specified by the state matrix `A` and input matrix `B`.

```
>> A = [0 1; 1 1];
```

```
>> B = [0; 1];
>> prob = set(prob, 'dynamics', A*x+B*u);
```

### 5.2.5   Setting the horizon length

**Property name:** `horizon`.
**Number of arguments:** 1.
**Default:** 0.
**Description:** Specifies the horizon length, a non-negative number. If set to 0, the horizon is free.
**Example**
Assume the POCP object `prob` has already been created. With the following command the horizon length is set to 1.

```
>> prob = set(prob, 'horizon', 1);
```

### 5.2.6   Setting the final cost

**Property name:** `fcost`.
**Number of arguments:** 1.
**Default:** 0.
**Description:** Specifies the final cost, a polynomial of GloptiPoly class `mpol` whose variables are elements of the state vector. If both the final cost and the integral cost are 0, then POCP minimizes the trace of the moment matrix of the occupation measure of the trajectory.
**Example**
Assume a POCP object `prob` has already been created and `x(1)` is one of the state variables. The following command sets the final cost to `x(1)^2`.

```
>> prob = set(prob, 'fcost', x(1)^2);
```

### 5.2.7   Setting the integral cost

**Property name:** `scost`.
**Number of arguments:** 1.
**Default:** 0.
**Description:** Specifies the integral cost, a polynomial of GloptiPoly class `mpol` whose variables are the time variable and elements of the state and input vectors. For minimum time optimal control problems, set `scost` to 1 and set `horizon` to 0. If both the final cost and the integral cost are 0, then POCP minimizes the trace of the moment matrix of the

```

occupation measure of the trajectory.

**Example**

Assume a POCP object `prob` has already been created and the state and input vectors are `x` and `u`, respectively. With the following command the integral cost is set to `x'*x+u'*u`.

```
>> prob = set(prob, 'scost', x'*x+u'*u);
```

### 5.2.8 Setting the dependence on time of the test functions

**Property name: `testtime`.**
**Number of arguments:** 1.
**Default:** empty.
**Description:** When the system dynamics does not depend on time and the horizon is free the test function does not depend on time. In the other cases the test function depends on time. However, this default behavior can be modified by setting the property `testtime`. When the argument is `true` (`false`) the test functions will (not) depend on time. To restore the default behavior, specify an empty (`[]`) argument.

**Example**

Assume a POCP object `prob` has already been defined. To make the test function depend also on time, use the following command

```
>> prob = set(prob, 'testtime', true);
```

### 5.2.9 Setting equality and inequality constraints on the initial condition

**Property name: `iconstraint`.**
**Number of arguments:** 1.
**Default:** empty.
**Description:** Specifices equality and inequality constraints on the initial condition, a column vector of GloptiPoly class `supcon`.

**Example**

Assume the POCP object `prob` has already been created and `x(1)` and `x(2)` are the state variables. With the following command, both variables are constrained to be less than or equal to 1.

```
>> prob = set(prob, iconstraint, [x(1)<=1; x(2)<=1]);
```

If the sum of `x(1)` and `x(2)` must be equal to 1, the following command should be used

```
>> prob = set(prob, iconstraint, x(1)+x(2)==1);
```

### 5.2.10 Setting the initial condition to a Dirac delta

**Property name:** `idirac`.
**Number of arguments:** 2 or 3.
**Default:** unspecified, considered as a problem unknown.
**Description:** This property should be set when one or more state variables take their value at time 0 accordingly to a distribution given by the sum of Dirac deltas. Two or three arguments can be specified. The first argument is a column vector containing the state variables involved. The second is a matrix whose columns contain the value of the variables (one column for each Dirac delta). The third argument (optional when the second argument has only one column) is a row vector specifying the probability of each Dirac delta.
**Example**
Assume a POCP object `prob` has already been created and `x(1)` and `x(2)` are the state variables. With the following command we set the initial condition to $x(0) = (1, 1)$

```
>> prob = set(prob, 'idirac', [x(1); x(2)], [1; 1]);
```

If the value at time 0 is $x(0) = (0, 1)$ with probability 0.8 and $x(0) = (1, 1)$ with probability 0.2, we should use

```
>> prob = set(prob, 'idirac', x, [0 1; 1 1]; [0.8 0.2]);
```

### 5.2.11 Setting the initial condition to a uniform distribution on a box

**Property name:** `iuniform`.
**Number of arguments:** 2.
**Default:** unspecified, considered as a problem unknown.
**Description:** This property should be set when one or more state varibles take their value at time 0 accordingly to a uniform distribution on a box. Two arguments must be specified. The first argument is a column vector containing the state variables involved. The second is a matrix with two columns and the number of rows equal to the length of the first argument, specifying the intervals.
**Example**
Assume a POCP object `prob` has already been created and `x(1)` and `x(2)` are the state variables. With the following command we set the value of `x(1)` to be uniformly distributed on the interval $[-1, \ 1]$ and the value of `x(2)` on the interval $[0, \ 2]$

```
>> prob = set(prob, 'iuniform', [x(1); x(2)], [-1 1; 0 2]);
```

The properties `idirac` and `iuniform` can also be used in the same problem, like in the following

```
>> prob = set(prob, 'iuniform', x(1), [-1 1]);
>> prob = set(prob, 'idirac', x(2), -1);
```

### 5.2.12 Setting equality and inequality constraints on the final condition

**Property name:** fconstraint.
**Number of arguments:** 1.
**Default:** empty.
**Description:** Specifies equality and inequality constraints on the state variables at the end of the horizon. The usage is the same as for the property iconstraint.

### 5.2.13 Setting the final condition to a Dirac delta

**Property name:** fdirac.
**Number of arguments:** 2 or 3.
**Default:** unspecified, considered as a problem unknown.
**Description:** This property should be set when one or more state variables take their value at the end of the horizon accordingly to a distribution given by the sum of Dirac deltas. The usage is the same as for the property idirac.

### 5.2.14 Setting equality and inequality constraints on the variables along the trajectory

**Property name:** tconstraint
**Number of arguments:** 1.
**Default:** empty.
**Description:** Specifies equality and inequality constraints on the system variables along the trajectory, a column vector of GloptiPoly class supcon.
**Example**
Assume a POCP object prob has already been created and x is the scalar state, u is the input, and t is the time variable. With the following command we set the constraints x>=0 and x+u+t<=1

```
>> prob = set(prob, 'tconstraint', [x>=0; x+u+t<=1]);
```

### 5.2.15 Setting integral constraints

**Property name:** sconstraint.
**Number of arguments:** 1.

**Default:** empty.
**Description:** Specifices constraints on the integral of some variables, a column vector of GloptiPoly class `momcon`.
**Example**
Assume a POCP object `prob` has already been created and `u` is the input variable. To enforce the integral constraint $\int_0^T u(t)^2 dt \leq 1$, use the following command

```
>> prob = set(prob, 'sconstraint', mom(u^2)<=1);
```

### 5.2.16 Some remarks on the usage of the `set` command

- The system variables (state vector, input vector, and time) must be set before they are used. Constraints on the initial state cannot be entered if the state vector has not been set. When more properties are specified with the same `set` command, the system variables should be specified first.

- Once the system variables (state vector, input vector, and time) have been specified, they cannot be modified any further.

- The current version of the software does not check for conflicts between different kinds of constraints. E.g., if the value at time 0 for a state variable is assigned by setting the property `idirac` and then the same variable appears in the constraints set by `iconstraint`, the behavior of POCP is unpredictable.

## 5.3 The `solvepocp` command

**Short description:** Builds and solves an SDP relaxation of a POCP.
**Syntax and usage:** When used in the form

```
>> [status, cost, measures] = solvepocp(prob, degree);
```

or, equivalently,

```
>> [status, cost, measures] = solvepocp(prob, 'mom', degree);
```

the command `solvepocp` builds and solves an SDP relaxation where moments of degree up to `degree` are used. The input argument `'mom'` stands for moments. The output parameters have the following meaning:

- When `status` is strictly negative, the SDP problem is infeasible or could not be solved. Otherwise, the SDP problem could be solved;

- `cost` is a lower bound on the objective function of the POCP if `status` is non-negative;

- `measures` is a structure containing the measures used by GloptiPoly to solve the relaxation of the POCP. Once the relaxation is solved, `measures` contains fields `measures.initial`, `measures.final`, and `measures.trajectory`. If the initial or final conditions have been assigned, the corresponding measures are empty (`[]`). To extract the moment matrix corresponding to one of the measures, use the GloptiPoly function `mmat` in combination with `double`. For example, to retrieve the moment matrix corresponding to the trajectory occupation measure, use the following command

  ```
  >> double(mmat(measures.trajectory));
  ```

It is also possible to call `solvepocp` by specifying the degree of the test function to be used when defining the moment problem:

```
>> [status, cost, measures] = solvepocp(prob, 'tf', degree);
```

The input argument `'tf'` stands for test function. As a rule of thumb, `degree` should be at least twice the degree of the dynamics. Increasing the value of `degree` improves the quality of the approximation, but it also increases the size of the SDP problem solved.

The command `solvepocp` can also be called by specifying four output parameters:

```
>> [status, cost, measures, vf] = solvepocp(prob, degree);
```

Using this syntax, a subsolution of the HJB equation is calculated and stored in `vf` (for value function). This approximation is a polynomial of GloptiPoly class `mpol`. If the option `'tf'` is used, `degree` corresponds to the degree of the subsolution of the HJB equation. The subsolution `vf` is computed by using the YALMIP toolbox, which must be installed properly.

# 6   First example

In this section we show how POCP can be used to design a suboptimal controller. Consider as in [1] the nonlinear system

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} x_2(t) - x_1(t)^3 + x_1(t)^2 \\ u(t) \end{bmatrix}.$$

The optimal control problem consists in driving to the origin the initial states from the set $(x_1, x_2) \in \mathcal{S} = [-1, 1] \times [-1, 1]$ by minimizing the cost functional $\int_0^T h(x(t), u(t))dt$ with

$$h(x(t), u(t)) = x_1(t)^2 + x_2(t)^2 + \frac{u(t)^2}{100}.$$

The terminal time $T$ is free.

To solve this problem and design a control law, we can define a POCP problem and use the command `solvepocp` with four output arguments. By setting the initial condition to be uniformly distributed on the box $\mathcal{S}$, the fourth output argument of `solvepocp` contains a polynomial subsolution of the HJB equation which approximates the value function along all the optimal trajectories starting from $\mathcal{S}$, see [1]. Therefore, a natural choice to derive a control law consists in solving the following minimization problem

$$\min_u \left[ \frac{\partial V_f(x, t)}{\partial t} + \nabla_x V_f(x) f(x, u) + h(x, u) \right], \tag{5}$$

where $V_f$ represents the approximated value function. Since the dynamics is affine in $u$ and $h()$ is quadratic in $u$, the global minimizer of (5) can be calculated by using first order optimality conditions:

$$u(x) := -50 \nabla_x \varphi(x) \begin{bmatrix} 0 & 1 \end{bmatrix}^T.$$

In the sequel we show a Matlab POCP script which calculates a polynomial approximation of the value function `vf` and defines the control law `u_x`.

```
% variable declaration
mpol x 2
mpol u

% problem definition
prob = pocp( ...
        'state', x, ...
        'input', u, ...
        'dynamics', [x(2)+x(1)^2-x(1)^3; u], ...
        'horizon', 0, ...
        'iuniform', x, [-1 1; -1 1], ...
        'fd', x, [0;0], ...
        'scost', x'*x+u^2/100);

% problem solved with test function up to degree 8
[status, cost, measures, vf] = solvepocp(prob, 'tf', 8);

% control law definition
u_x = -50 * diff(vf, x)*[0; 1];
```

The value of `u_x` can be easily evaluated by using the GloptiPoly commands `assign` and `double`. For example, to obtain a value for the control at $x = (0.5, \ 0.5)$, the following commands can be used.

```
>> assign([x; u], [0.5 0.5 0]');
>> controlvalue = double(u_x);
```

Notice that when we assign the value of x we must assign also the value of u (any value can be used). For more information on the commands `assign` and `double`, see the documentation of GloptiPoly.

Figure 1 shows some trajectories starting from $\mathcal{S}$ and obtained by simulating the system with the calculated control law.



Figure 1: Trajectories obtained by applying the control law `u_x`.

# 7    Second example

This example is taken from [5, Example 3.27]. For the nonlinear system

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} -x_1(t)^3 + x_1(t)u(t) \\ u(t) \end{bmatrix}$$

and the cost functional $\int_0^\infty (x_2(t)^2 + u(t)^2)dt$, the solution of the HJB equation is $x_2(t)^2$, which is only positive semidefinite. Since the solution is polynomial, we can show with this example that the solution can be found exactly (and only using test functions up to degree 2):

```
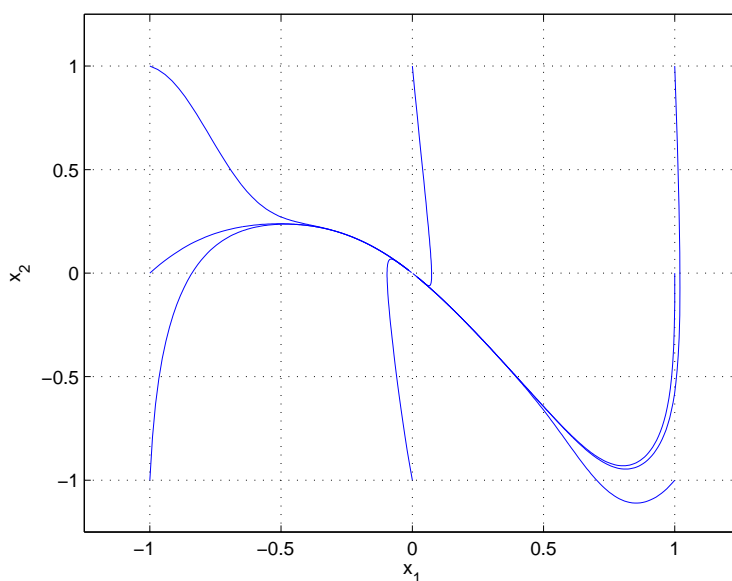mpol x 2
```

```
mpol u

p=pocp();
p=set(p,'state',x);
p=set(p,'input',u);
p=set(p,'dynamics',[-x(1)^3+x(1)*u; u]);
p=set(p,'scost',x(2)^2+u^2);
p=set(p,'idirac',x, [1; 1]);
p=set(p,'fdirac',x, [0; 0]);
p=set(p,'tconst', [x>=-1.1; x<=1.1]);

[status,cost,measures,vf] = ...
 solvepocp(p,'tf', 2);

disp('Subsolution of HJB equation=');
vf
```

Running the above script we obtain the following output:

```
Scalar polynomial
x(2)^2
```

# 8  Variable scaling

To improve the numerical behavior of the SDP solver used by POCP, the variables (state, input and time) could be scaled. Scaling can be achieved by using the GloptiPoly command `scale`. For example, to substitute the variable x with 2*x, use the following commands

```
>> mpol x
>> scale(x, 2)
```

Scaling changes only the internal representation of the variables. From the user point of view nothing changes, and there is no need to scale the result of the optimization problem. For more information about the `scale` command, see the documentation of GloptiPoly. If possible, all the variables should be scaled within the interval $[-1, +1]$.

# 9  Some ideas for new features

We end this guide by listing ideas for improvements and new features of POCP:

- To further extend the class of continuous-time problems considered, more flexibility on the initial and final time (currently 0 and $T$) could be permitted. If the cost

$$J(t_i, t_f, x(0), u(\cdot)) = \int_{t_i}^{t_f} h(t, x(t), u(t))dt + H(x(t_f)). \tag{6}$$

is considered, the problem

$$\begin{aligned} \min_{u(\cdot)} \quad & \int J(t_i, T, x_i, u(x_i, t))d\mu_i(t_i, x_i) \\ & (t, x(t), u(t)) \in \mathcal{C}_T \\ & x(T) \in \mathcal{C}_F \end{aligned} \tag{7}$$

could be modeled with POCP, with $\mu_i$ an assigned probability measure on $\mathbb{R} \times \mathbb{R}^n$. Setting $\mu_i$ to be a uniform measure on the set $\{(t, x) : t \in [0, 1], \; x_j \in [-1, -1] \; \forall j = 1, \ldots, n\}$, we could obtain a subsolution of the HJB equation which gives a good approximation of the value function on a box centered at the origin of the state space and for all the values of $t$ in the interval $[0, 1]$, see [1].

- Version 3.3 of GloptiPoly cannot be used to retrieve information on the dual variables of the moment problem. As a workaround to calculate a subsolution of the HJB equation, POCP currently uses GloptiPoly in combination with YALMIP. If GloptiPoly will include this feature in some future version, POCP will not require YALMIP.

- GloptiPoly does not support currently the assignment of a measure (e.g., it cannot be specified that a variable is uniformly distributed on an interval). If GloptiPoly will include this feature in some future version, the POCP code could be significantly simplified.

# References

[1] D. Henrion, J. B. Lasserre, C. Savorgnan, *Nonlinear optimal control synthesis via occupation measures*, Proc. IEEE Conf. Decision and Control, 2008.

[2] J. B. Lasserre, D. Henrion, C. Prieur, E. Trélat, *Nonlinear optimal control via occupation measures and LMI relaxations*, SIAM J. Control Opt., 47(4):1643-1666, 2008.

[3] D. Henrion, J. B. Lasserre, J. Löfberg, *GloptiPoly 3: moments, optimization and semidefinite programming*, LAAS-CNRS Research Report 07536, June 2007.

[4] J. Löfberg, *YALMIP: a toolbox for modeling and optimization in Matlab*, Proc. IEEE Symp. CACSD, 2004.

[5] R. Sepulchre, M. Janković, P. Kokotović. *Constructive Nonlinear Control*, Springer Verlag, 1997.