Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

RACT: Randomized Algorithms Control Toolbox for MATLAB [1]

Andrey Tremba * **Giuseppe Calafiore** ** **Fabrizio Dabbene** ***,2
Elena Gryazina * **Boris Polyak** * **Pavel Shcherbakov** *
Roberto Tempo ***

* *Institute of Control Sciencies, Moscow, Russia*
** *DAUIN Politecnico di Torino, Italy*
*** *IEIIT-CNR Politecnico di Torino, Italy*

Abstract: This paper introduces a new Matlab package, Ract, aimed at solving a class of probabilistic analysis and synthesis problems arising in control. The package offers a convenient way for defining various types of structured uncertainties as well as formulating and analyzing the ensuing robustness analysis tasks from a probabilistic point of view. It also provides a full-featured framework for LMI-formulated probabilistic synthesis problems, which includes sequential probabilistic methods as well as scenario methods for robust design. The Ract package is freely available at `http://ract.sourceforge.net`, and only requires the Yalmip toolbox to be installed in the Matlab environment.

Keywords: Probabilistic robustness; Randomized methods; Robust controller synthesis; LMIs; Uncertainty description.

1. INTRODUCTION

Probabilistic and randomized techniques for analysis of uncertain systems and design of robustly performing control systems have attracted considerable interest in recent years, and a significant amount of theoretical and algorithmic results have appeared in the literature. A rather comprehensive source of pointers to available results in this field is given in the books Tempo et al. (2005) and Calafiore and Dabbene (2006) and in the surveys Calafiore et al. (2007); Calafiore and Dabbene (2007a); Tempo and Ishii (2007).

The starting idea in the probabilistic approach to the analysis of uncertain systems is to characterize the uncertain parameters as random variables, and then to evaluate the system performance in terms of probabilities. Analogously, probabilistic synthesis is aimed at determining the design parameters so that certain desired levels of performance are attained with high probability. This probabilistic approach is complementary to the mainstream methods in robust control, which seek worst-case performance guarantees and consider the uncertainties as deterministic unknown-but-bounded quantities. These latter methods may have limitations due to conservatism and computational complexity in real-world situations where a large number of uncertain parameters enter the system description in a possibly nonlinear way.
In order to diffuse and to make these techniques easily accessible to the interested researchers, we made an effort to unify them into a coherent set of Matlab routines that constitute the Ract toolbox. This paper presents, by means of examples, the main features and functionalities available in this toolbox. Detailed instructions are provided in the User's Manual (Tremba et al. (2007)). The Ract toolbox contains only text `m`-files and its installation simply amounts to adding some specific folders to the Matlab path. The freely available Yalmip toolbox (Lofberg, 2004) and an SDP (sedmidefinite programming) solver also need to be installed in the Matlab environment.

[1] This work was supported by a bilateral international project funded by Consiglio Nazionale delle Ricerche (CNR) and Russian Academy of Sciences (RAS). The support of an FIRB grant from the Italian ministry of education and research is also acknowledged.
[2] Corresponding author (`fabrizio.dabbene@polito.it`).

2. DEALING WITH UNCERTAINTY

Ract provides easy and efficient tools for defining and sampling a random *uncertain object.* An uncertain object is a particular object (e.g., transfer function, polynomial or LMI) which depends in a complex and structured way on a set of uncertain parameters, which are represented by random variables with given distribution, referred to as *basic uncertainties.* A natural way to represent an uncertain object is to define the uncertainties that compose it and the structure of the object itself separately. For both analysis and synthesis problems, Ract requires to generate a (usually large) number of samples of the uncertain object. To this end, the following functional paradigm is adopted. First, Ract provides a series of basic uncertainties (scalar, vector and matrix ones with different given distributions) by means of a special Matlab class. Then, the user defines an `m`-file for each uncertain object (or uncertain problem), following some intuitive composition rules. The structure of these "user-defined" `m`-files (user-function) follows a clear template form. Once these user-function files are defined, the desired randomized algorithm can be utilized by a simple call to the specific Ract routine, with the *handle* [3] to the user-function as argument. This functional approach appears to be quite flexible and provides a nice compromise between speed of execution on one side and usability on the other; the range spans from writing a Randomized Algorithm (RA) code from scratch for every single problem to using a symbolic toolbox and uncertainty substitution (perhaps more user-friendly, but computationally an extremely slow alternative).

2.1 Basic random variable definition and sampling

The basic uncertainties are handled by means of the new MATLAB class `ubase`. For example, a uniformly distributed vector $q_\Delta \in \mathbb{R}^2$, bounded in an ℓ_p-ball with parameter $p = 2$, $\|q_\Delta - q_0\|_2 \le 0.5$, $q_0 = (1, 2)^T$ is constructed using the command

```
>> q=ubase('q_delta','real_vector_uniform',...
   'nominal', [1; 2], 'p_norm', 2, 'rho', 0.5)
```

In the syntax of `ubase`, the variable name (or a small description of it) is listed first, then a string with the probability distribution, and finally a series of paired distribution parameter names and corresponding values. For the previous example, these parameters are respectively: a column [4] nominal vector q_0, the value of the ℓ_p–norm parameter p and a scaling factor `rho`.

The full list of these commands is (`see >>help ubase` for more details) is given below:

- `real_scalar_uniform` — real scalar random variable (r.v.) uniformly distributed in an interval;
- `complex_scalar_uniform` — complex scalar r.v. uniformly distributed in a disk in the complex plane;
- `real_scalar_gaussian` — real scalar r.v. with Gaussian distribution;
- `real_vector_uniform` — real random vector uniformly distributed in the l_p-ball;
- `complex_vector_uniform` — complex random vector uniformly distributed in the l_p-ball;
- `real_vector_gaussian` — real random vector with Gaussian distribution;
- `real_stable_discrete_poly_uniform` — the vector of coefficients of a monic discrete stable polynomial uniformly distributed in the coefficient space;
- `real_matrix_uniform` — real random matrix uniformly distributed in induced norm;
- `complex_matrix_uniform` — complex random matrix uniformly distributed in induced norm;
- `real_matrix_uniform_elem` — real, uniformly distributed element-wise interval matrix.

For the example at the beginning of this section, the routine for generating a given number of samples (say, four) of a basic uncertainty is `ubasesample`, having the form

```
>> q_samples = ubasesample(q, 4)

q_samples =
    0.5385    0.9808    1.0803    1.3677
    1.7268    1.9473    1.8840    2.1518
```

[3] A *function handle* is a MATLAB value that provides a way of calling a function indirectly. For instance, the handle to a function named `myfunc.m` can be created using the syntax `@myfunc`.

[4] RACT treats all uncertain and nominal vectors as column vectors.

These samples are packed into a matrix, column-wise along the second dimension. Scalars and matrices are treated using the same formalism: scalar samples are packed into a column vector and matrix samples are packed along the 3rd dimension (the k-th sample casts as `M_k = M_samples(:,:,k)`). The nominal value is extracted by setting the second argument equal to zero.

Example 1. (Uncertain object construction and sampling). As a simple example of an uncertain object, we consider the following uncertain polynomial:

$$\begin{aligned} p(s, q_1, q_2) = {} & s^4 + (4 + q_2)s^3 + (6 + 3q_2)s^2 + \\ & +(5 + q_1 + 3q_2)s + 2 + q_1 + q_2, \end{aligned} \tag{1}$$

with q_1 being uniformly distributed in the interval $|q_1| \le 1.5$, and the second uncertainty q_2 having Gaussian distribution with mean 0.15 and variance 0.25.

The `m`-file describing the uncertain object consists of three main sections (the gray-colored lines constitute the toolbox template and should not be edited):

```
function Out = ex_unc_poly(Init, UncVar)
if Init
  UncVar = cell(0);
  %% Section 1 - Define basic uncertainties
  UncVar{1} = ubase('q1', 'real_scalar_uniform',
                    'nominal', 0, 'range', 1.5);
  UncVar{2} = ubase('q2', 'real_scalar_gaussian',
                    'mean', 0.15, 'std', 0.5);

  %% End of Section 1
  Out = UncVar;
  return;
end
%% Section 2 - Extract samples
%%             of basic uncertainties
q1 = UncVar{1};
q2 = UncVar{2};
%% End of Section 2
%% Section 3 - Compose a sample of
%%             uncertain polynomial:
Out{1} = [1, 4+q2, 6+3*q2, 5+q1+3*q2, 2+q1+q2];
%% End of Section 3
```

The first section is the initialization part, where both uncertain variables q_1, q_2 are defined, together with their distribution parameters. The second section shows sample extraction: the cell array `UncVar` is passed on to the function as argument, and contains samples of the two uncertain variables defined above. In this section, the sample values are assigned back to their respective variables. Finally, a sample of the uncertain polynomial is returned in the third section. To summarize, suppose that the `m`-file above has been saved as `ex_unc_poly.m`. Then, sampling the uncertain object (in this case, the polynomial $p(s, q_1, q_2)$) is performed in a one-line command, with the number of samples as second argument

```
>> poly_samples = uevaluate(@ex_unc_poly, 2)

poly_samples =
    [1x5 double]
    [1x5 double]
```

The result is a one-dimensional cell array, with k-th sample `poly_samples{k}`. $\Diamond$

3. RANDOMIZED ALGORITHMS FOR ROBUSTNESS ANALYSIS

In this section we briefly discuss the RACT routines for probabilistic robustness analysis.

Formally, we let $\Delta \in \mathbb{D}$ represent the random uncertainty affecting the system, where $\mathbb{D}$ is the support of the random object Δ (for instance, $\mathbb{D}$ can be the space of ℓ-dimensional real vectors, or the space of block-structured matrices with a norm bound) and denote with $f_\Delta(\Delta)$ the probability density function (pdf) of Δ. Let further $J(\Delta) : \mathbb{D} \to \mathbb{R}$ be a *performance function* for the uncertain system, i.e. a function measuring the performance of the system for a given value of Δ. For instance $J(\Delta)$ can be the $\mathcal{H}_2$ or the $\mathcal{H}_\infty$ norm of the system.

Classical robustness analysis deals with the question if a given performance requirement $J(\Delta) \le \gamma$ holds for the whole uncertain system family. In other words, a "yes-or-no" answer is required. In contrast, in the probabilistic robustness approach, the system uncertainty is adopted to be random, and the fundamental concept of probability of performance is introduced as

$$\mathrm{Prob}\left\{\Delta \in \mathbb{D} \,:\, J(\Delta) \le \gamma\right\}, \tag{2}$$

where the probability is measured according to the underlying uncertainty distribution. Hence, the classical deterministic requirement is softened and formulated as one of the two following tasks:
1. *Performance verification*: What is the probability that the property of interest holds?
2. *Worst-case performance*: For which performance level γ, the probability of performance holds with a desired probability?

In general, the exact computation of probability (2) turns out to be even more complicated than solving its deterministic counterpart. However, this probability can be estimated with given confidence and accuracy by means of randomized algorithms, as explained next.

3.1 RA for probabilistic performance verification

First, we specify the characteristics that a RA for probabilistic performance verification should comply with.

Definition 1. (RA for performance verification).
Let $\epsilon \in (0,1)$, $\delta \in (0,1)$ be assigned probability levels. Given a performance level γ, the RA should return with probability $1-\delta$ an estimate p_{est} of the probability of performance

$$p \doteq \mathrm{Prob}\{J(\Delta) \le \gamma\},$$

which is within ϵ-accuracy from p, i.e.

$$\mathrm{Prob}\{|p - p_{\text{est}}| \le \epsilon\} \ge 1-\delta.$$

The estimate p_{est} should be constructed based on a finite number N of random samples of Δ.

Notice that a simple RA for performance verification is directly constructed by means of the classical Monte Carlo method as follows:

$$p_{\text{est}} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[J(\Delta^{(i)}) \le \gamma],$$

where $N = \left\lceil \frac{1}{2\epsilon^2} \ln \frac{2}{\delta} \right\rceil$ is dictated by the Chernoff bound (Chernoff (1952)), $\Delta^{(i)}$ are independent identically distributed (iid) samples, and the indicator function $\mathbb{I}[\cdot] = 1$ when its argument is true, and zero otherwise.

This algorithm is implemented in the RACT routine `perfver`, and its use in demonstrated in the following example on stability of an uncertain polynomial.

Example 2. (Stability analysis of an uncertain polynomial). Consider the uncertain polynomial

$$p(s,q) = s^3 + q_1 s^2 + q_2 q_3 s + q_2, \tag{3}$$
$$|1-q_1| \le 1.1, |1-q_2| \le 1, |1-q_3| \le 0.5,$$

where q is uniform over its domain. Suppose we wish to check if the polynomial is robustly stable in a probabilistic sense, i.e., we are interested in estimating the probability of stability.

To utilize the RACT framework, an m-file `sample_poly.m` containing the description of the uncertain polynomial (3) is created, following the three-section template previously described. Then, to compute an estimate of the probability of stability of the uncertain polynomial $p(s,q)$, with accuracy level $\epsilon = 0.01$ and confidence $\delta = 0.001$, we simply invoke the command `perfver`

```
>> p_est = perfver(@sample_poly, 'hurwitz',
                   0.01, 0.001)

Number of samples by Chernoff bound is: 38005
With probability 0.999,  |p - 0.45973| <= 0.01
ans =
    0.4597
```

In the argument list of `perfver`, first we have the handle to the function defining the uncertain object (polynomial in this case), then the handle to the property indicator function (or built-in name of such a function, other options are for instance `'schur'` for Schur stability test, `'posdef'` for positive definitness etc.), and finally the probability levels ϵ and δ. ◊

3.2 RA for probabilistic worst-case performance

The second robustness analysis problem considered is assessing the worst-case performance level of the system. In this case, we shall consider a RA that enjoys the following features.

Definition 2. (RA for worst-case performance).
Let $p^* \in (0,1)$, $\delta \in (0,1)$ be assigned probability levels. The RA should return with probability $1-\delta$ a performance level γ_{est} such that

$$\mathrm{Prob}\{J(\Delta) \le \gamma_{\text{est}}\} \ge p^*.$$

The performance level γ_{est} should be constructed based on a finite number N of random samples of Δ.

In words, a RA for probabilistic worst-case performance should determine a performance level γ_{est} which is guaranteed for most of the uncertainty instances. This can be obtained via the empirical maximum

$$\gamma_{\text{est}} = \max_{i=1,\dots,N} J(\Delta^{(i)}),$$

where $N = \left\lceil \frac{\ln 1/\delta}{\ln 1/p^*} \right\rceil$ is given by the so-called "log-over-log" bound, which provides the number of required iid samples (Tempo et al. (1997)).

This algorithm is implemented in the RACT function `perfwc`, and its use is perfectly analogous of that of `perfver`: the user has to create two functions: the first one containing the definition of the uncertain problem, and the second one defining the performance criterion. For numerical examples and implementation details see Tremba et al. (2007). In the argument of the function `perfwc`, first we list the handle to the uncertain object function, then the handle to the criterion, the probability p^*, and, finally, the confidence level δ.

4. RANDOMIZED ALGORITHMS FOR SYSTEM DESIGN

The application domain of the probabilistic approach is not limited to analysis problems, but has been extended to deal with the synthesis of robustly performing control systems. This is actually the most interesting and promising area of application of randomized techniques. Some of the methods recently proposed in the literature have been implemented in RACT. In particular, the toolbox is focused on design techniques based on the interplay of random sampling in the uncertainty space, and deterministic convex optimization in the design parameter space.

Formally, let $\theta \in \Theta \subseteq \mathbb{R}^{n_\theta}$ denote the vector of design variables, and let $\Delta \in \mathbb{D}$ denote the uncertain parameters. In Section 3.1, the performance function is denoted by $J(\Delta)$, and the performance objective is to verify the inequality $J(\Delta) \leq \gamma$ for a desired performance level γ. If the system depends on a design parameter θ, the performance function should also depend on θ, that is, $J = J(\Delta, \theta)$. For generality and ease of notation, in this section we denote all the design and performance constraints by the design inequality $f(\Delta, \theta) \leq 0$, where $f(\Delta, \theta) : \mathbb{D} \times \Theta \to \mathbb{R}$ is a scalar-valued function[5] related to the system with design parameters θ. For instance, if the design objective is to determine θ such that $J(\Delta, \theta) \leq \gamma$, we simply take $f(\Delta, \theta) = J(\Delta, \theta) - \gamma$.

The main assumption of this section is that the function $f(\Delta, \theta)$ should be *convex* in θ for all $\Delta \in \mathbb{D}$. Another technical assumption is that the solution set should contain a full-dimensional ball of radius r (e.g., see Calafiore and Polyak (2001)). The design vector θ such that the inequality $f(\Delta, \theta) \leq 0$ is satisfied "for most" (in a probabilistic sense) of the outcomes of Δ is called a *probabilistic robust* design.

We next define the two problems that we aim to solve:
1) *find* a probabilistic robust design (i.e., solve a feasibility problem)
2) *optimize* a probabilistic feasible design (that is, solve an optimization problem).

[5] We remark that considering scalar-valued constraint functions is without loss of generality, since multiple constraints $f_1(\Delta, \theta) \leq 0, \ldots, f_{n_f}(\Delta, \theta) \leq 0$ can be reduced to a single scalar-valued constraint by setting $f(\Delta, \theta) \doteq \max_{i=1,\ldots,n_f} f_i(\Delta, \theta)$.

The randomized algorithms we discuss in this section provide a numerically viable way to compute approximate solutions for the two problems above.

4.1 RA for probabilistic robust design

First, we specify the features that a randomized algorithm for probabilistic robust design should comply with.

Definition 3. (RA for probabilistic robust design).
Let $p^* \in (0, 1)$ be a probability level. The RA should return with probability $1 - \delta$ a design parameter $\theta_{\rm pr} \in \Theta$ such that

$$\mathrm{Prob}\{\Delta \in \mathbb{D} : f(\theta_{\rm pr}, \Delta) < 0\} \geq p^*. \qquad (4)$$

The design parameter $\theta_{\rm pr}$ should be constructed based on a finite number of random samples of Δ.

The following meta-algorithm implements this RA.

Algorithm 1. (Sequential RA for robust design)
Given $p^*, \delta \in (0, 1)$, returns with probability at least $1 - \delta$ a design vector $\theta_{\rm pr}$ such that (4) holds.

```
(1) Initialization.
      ▷ Set $k = 0$ and choose $\theta_0 \in \Theta$;
      ▷ Determine the sample-size function $N(k)$
            $N(k) = N_{ss}(p^*, \delta, k) \doteq \left\lceil \frac{\log \frac{\pi^2 (k+1)^2}{6\delta}}{\log \frac{1}{p^*}} \right\rceil$;    (5)
(2) Probabilistic oracle.
      ▷ For $i = 0$ to $N(k)$
            · Draw a sample $\Delta^{(i)}$
            · If $f(\Delta^{(i)}, \theta_k) > 0$ set $\Delta_k = \Delta^{(i)}$, feas=false
              and goto 3.
      ▷ End;
      ▷ Set feas=true, return $\theta_{\rm pr} = \theta_k$ and Exit;
(3) Update.
      ▷ Update $\theta_{k+1}$ by $(\theta_k, \Delta_k)$.
      ▷ Set $k = k + 1$ and goto 2.
```

Notice that the core of the above algorithm is the *Probabilistic Oracle*. A call to the probabilistic oracle may have two possible outcomes. If the `for` loop is run up to $i = N(k)$, the oracle exits returning `feas=true`, meaning that the query point θ_k passed the feasibility test on the $N(k)$ trials. In this latter case the point θ_k is labeled as *probabilistic feasible solution*: in fact, in Oishi (2007) it is shown that, if the sample size function $N(k)$ is chosen as in (5) then, with probability greater than $1 - \delta$, the returned solution $\theta_{\rm pr}$ satisfies (4).

Otherwise, if the `for` loop in the oracle is interrupted at some $i < N(k)$, then the probabilistic oracle exits returning `feas=false`, along with a "certificate of violation" Δ_k, $f(\theta_k, \Delta_k) > 0$.

Then, the candidate solution θ_k is updated by means of one of three algorithms: gradient, ellipsoid or cutting plane. The user can check any problem against all of these methods, choosing the most appropriate ones, as all of them have advantages and drawbacks Calafiore and Polyak (2001); Polyak and Tempo (2001); Kanev et al. (2003); Oishi (2007); Calafiore and Dabbene (2007b).

However, it should be remarked that implementing these three algorithms in their general formulation would need the knowledge of a (sub)gradient of the function f with

respect to θ (for fixed Δ). For this reason, and stemming from the consideration that many problems in control can be rewritten in linear matrix inequality (LMI) form, in RACT we have chosen LMIs as our main tool. That is, we consider robust feasibility of uncertain problems expressed in the form of a system of LMI constraints:
Find θ such that

$$F^{(j)}(\Delta,\theta) \preceq 0 \quad \forall \Delta \in \mathbb{D}, \quad j=1,\ldots,\ell,$$

where

$$F^{(j)}(\Delta,\theta) = F_0^{(j)}(\Delta) + \sum_{i=1}^{n} \theta_i F_i^{(j)}(\Delta), \tag{6}$$

and $F_i^{(j)}(\Delta)$, $i=0,\ldots,n$, are symmetric $m \times m$ real matrices which depend in a generic and possibly nonlinear way on the uncertainty $\Delta \in \mathbb{D}$. To rewrite this problem in our scalar-function framework, we set

$$f(\Delta,\theta) \doteq \max_j \lambda_{\max}(F^{(j)}(\Delta,\theta)). \tag{7}$$

A subgradient of the function $f(\Delta,\theta) = \lambda_{\max}(F(\Delta,\theta))$ at $\theta = \theta_k$ is readily computable as

$$g_\Delta(\theta_k) = \left[\, \xi_{\max}{}^T F_1(\Delta)\xi_{\max} \; \cdots \; \xi_{\max}{}^T F_n(\Delta)\xi_{\max} \,\right]^T,$$

where $\xi_{\max}$ is an eigenvector associated with the largest eigenvalue of $F(\Delta,\theta_k)$.

In order to handle uncertain LMIs within RACT, the YALMIP toolbox is used and should be installed beforehand. With this choice, there is no need to recast the original LMI problem in the general form (6) and we can take advantage of the ability of YALMIP to parse generic LMI problems. Moreover, there is no need to perform manually gradient calculations. The RA for probabilistic robust LMI is implemented in the RACT command `feaslmi` which is described in the next example.

Example 3. (Probabilistic robust LQR). Consider an example of uncertain LQR (linear quadratic regulator) problem. For the linear state space system

$$\dot{x}(t) = A(\Delta)x(t) + Bu(t),$$

a linear quadratically stabilizing controller is given by $u(t) = -B^T Q^{-1} x(t)$, where Q is a solution of the Lyapunov equation

$$A(\Delta)Q + QA(\Delta)^T - 2BB^T \prec 0, Q = Q^T \succ 0, \forall \Delta \in \mathbb{D}.$$

In particular, we consider the following state space equation describing the longitudinal dynamics of an aircraft

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & L_p & L_\beta & 0.78 \\ g/V & 0 & Y_\beta & -1 \\ N_{\dot\beta} & -0.042 & 2.601 + N_{\dot\beta} Y_\beta & -0.29 - N_{\dot\beta} \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 \\ 0 & -3.91 \\ 0.035 & 0 \\ -2.53 & 0.31 \end{bmatrix} u(t),$$

where each of the parameters $L_p = -2.93$, $L_\beta = -4.75$, $g/V = 0.086$, $Y_\beta = -0.11$, $N_{\dot\beta} = 0.1$, is perturbed by a relative uncertainty of 15%.

The template `feas_air` for stating this problem is very similar to the template for an uncertain object described in Example 1 (we notice that in this case the uncertain object is indeed an LMI).

```
function Out = feas_air(Init, UncVar)
%% Section 1 - declaration of sdp variables
persistent Q;
%% End of Section 1
switch Init
  case 'sdpvars'
    Out = cell(0);
%% Section 2 - sdpvar definition
    Q = sdpvar(4);
    Out{1} = Q;
%% End of Section 2
  case {1, 'init'}
  UncVar = cell(0);
%% Section 3 - basic uncertainties definition
  L_p_0 = -2.93; L_beta_0 = -4.75; g_over_V_0 = 0.086;
  Y_beta_0 = -0.11; N_betadot_0 = 0.1;
  UncVar{1} = ubase('L_p', 'real_scalar_uniform', ...
                  'nominal', L_p_0, 'range', 0.15*L_p_0);
  UncVar{2} = ubase('L_beta', 'real_scalar_uniform', ...
              'nominal', L_beta_0, 'range', 0.15*L_beta_0);
  UncVar{3} = ubase('g_over_V', 'real_scalar_uniform', ...
          'nominal', g_over_V_0, 'range', 0.15*g_over_V_0);
  UncVar{4} = ubase('Y_beta', 'real_scalar_uniform', ...
              'nominal', Y_beta_0, 'range', 0.15*Y_beta_0);
  UncVar{5} = ubase('N_betadot', 'real_scalar_uniform', ...
        'nominal', N_betadot_0, 'range', 0.15*N_betadot_0);
%% End of Section 3
  Out = UncVar;
  case {0, 'lmi'}
%% Section 4 - extract basic uncertainties samples
  L_p = UncVar{1};
  L_beta = UncVar{2};
  g_over_V = UncVar{3};
  Y_beta = UncVar{4};
  N_betadot = UncVar{5};
%% End of Section 4
%% Section 5 - construct matrices
  A =[...
     0,   1,    0,    0;...
     0,  L_p, L_beta, 0.78;...
     g_over_V, 0, Y_beta, -1; ...
     N_betadot*g_over_V, -0.042,
     2.601 + N_betadot*Y_beta, -0.29 - N_betadot];
  B = [ 0,      0; ...
        0,     -3.91; ...
        0.035,  0; ...
       -2.53,   0.31];
%% End of Section 5
%% Section 6 - construct an LMI
  F1 = set(Q > 0);
  F2 = set(A*Q + Q*A.' - 2*B*B.' < 0);
  Out = F1 + F2;
%% End of Section 6
end
```

The purpose of the different sections in the template are similar to those of the user-function for defining uncertain objects, with the additional declaration and definition of the design variable `Q`. All design variables are defined through the YALMIP special type `sdpvar`. In our specific case, in Section 2 of the template function, a symmetric 4×4 matrix is defined (see Lofberg (2004) for other options), and in Section 6, a sample of the Lyapunov equation is constructed using the standard YALMIP syntax.

This file is sufficient to describe the LQR problem. To perform optimization, we need to set an initial candidate solution Q_0 and let RACT know which solution method we choose: (`'grad'`, `'ell'`, `'cutplane'` for stochastic gradient, ellipsoid and cutting plane methods, respectively), and (if necessary) set other RA parameters:

```
>> Q0 = {zeros(4,4)};
>> grad_opt.method = 'grad';
>> grad_opt.Nout = 1000;
>> grad_opt.pstar = 0.999;
>> grad_opt.r = 1e-3;
>> grad_opt.delta = 1e-2;
```

where `pstar` and `delta` are the desired probability levels, `Nout` is the maximum number of iterations to claim infeasibility of a problem and `r` is the radius of a ball inscribed in the feasibility set.

Then, a RA for feasibility design runs in a single line as

```
>> Q_grad = feaslmi(@feas_air, Q0, grad_opt);
```

4.2 RA for probabilistic optimal design

Second, we consider the problem of optimizing a linear function of the design parameter subject to probabilistic robust feasibility constraints.

Definition 4. (RA for optimal design).
Let $p^* \in (0,1)$ be a probability level. Given an objective vector $c \in \mathbb{R}^{n_\theta}$, the RA should return a design parameter $\theta_{\rm po} \in \Theta$ that minimizes the objective $c^T\theta$, while satisfying the constraint

$$\text{Prob}\{\Delta \in \mathbb{D} : f(\Delta, \theta_{\rm po}) < 0\} \geq p^*$$

with probability larger than $1 - \delta$. The design parameter $\theta_{\rm po}$ should be constructed based on a finite number N of random samples of Δ.

The scenario techniques (see Calafiore and Campi (2006) for details) provide a simple and theoretically sound way to design an algorithm that complies with Definition 4. The idea is to gather a sufficient number of uncertainty samples and optimize over these representatives.

Algorithm 2. (Scenario RA for optimal design).
Given $p^*, \delta \in (0,1)$ and an objective vector $c \in \mathbb{R}^{n_\theta}$, returns a design vector $\theta_{\rm po}$ such that the objective $c^T\theta$ is minimized while the constraint

$$\text{Prob}\{\Delta \in \mathbb{D} : f(\Delta, \theta) \leq 0\} \geq p^* \tag{8}$$

is satisfied with probability at least $1 - \delta$.

(1) Compute N as the smallest integer such that
$$\binom{N}{n_\theta}(p^*)^{N-n_\theta} = \delta;$$
(2) Draw N IID samples $\Delta^{(1)}, \ldots, \Delta^{(N)}$;
(3) Solve the scenario problem
$$\theta_{\rm po} = \arg\min_{\theta\in\Theta} c^T\theta \quad \text{subject to}$$
$$f(\Delta^{(i)}, \theta) \leq 0, \quad i = 1, \ldots, N. \tag{9}$$

We notice that the scenario problem (9) is a standard convex optimization problem with a finite number of constraints and therefore it is efficiently solvable in many specific cases of interest in control. This RA has been implemented in RACT in the LMI setup. The RACT command is `scenlmi`; its use is very similar to the one of `feaslmi`, see Tremba et al. (2007) for further details and examples.

5. CONCLUSION

A new MATLAB toolbox for probabilistic robustness analysis and probabilistic control design is presented in this paper. This toolbox provides convenient uncertain object manipulation and implementation of randomized methods using state-of-the-art theoretical and algorithmic results. The two main features of the package are a functional approach with `m`-file templates, and a definition of design problems in generic LMI format using the widely used YALMIP syntax. This first release of the toolbox provides an easy-to-use interface to current randomized algorithms for control and is intended to be used by researchers, engineers and students interested in robust control, uncertain systems and optimization. The package can be freely downloaded from `http://ract.sourceforge.net`.

REFERENCES

G. Calafiore and M.C. Campi. The scenario approach to robust control design. *Trans. Aut. Contr.*, 51(5):742–753, 2006.

G. Calafiore and F. Dabbene. *Probabilistic and Randomized Methods for Design under Uncertainty.* (Edited book) Springer-Verlag, London, 2006.

G. Calafiore and B.T. Polyak. Stochastic algorithms for exact and approximate feasibility of robust LMIs. *Trans. Aut. Contr.*, 46:1755–1759, 2001.

G. Calafiore, F. Dabbene, and R. Tempo. A survey of randomized algorithms for control synthesis and performance verification. *J. Compl.*, 23(3):301–316, 2007.

G.C. Calafiore and F. Dabbene. Probabilistic methods in control: A tutorial. In *Proc. ACC*, 2007a.

G.C. Calafiore and F. Dabbene. Probabilistic analytic center cutting plane method for feasibility of uncertain LMIs. *Automatica*, 43:2022–2033, 2007b.

H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

S. Kanev, B. De Schutter, and M. Verhaegen. An ellipsoid algorithm for probabilistic robust controller design. *Sys. Cont. Lett.*, 49:365–375, 2003.

J. Lofberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proc. CACSD*, 2004.

Y. Oishi. Polynomial-time algorithms for probabilistic solutions of parameter-dependent linear matrix inequalities. *Automatica*, 43(3):538–545, 2007.

B.T. Polyak and R. Tempo. Probabilistic robust design with linear quadratic regulators. *Sys. Cont. Lett.*, 43: 343–353, 2001.

R. Tempo and H. Ishii. Monte Carlo and Las Vegas randomized algorithms for systems and control: An introduction. *Eur. J. Contr.*, 13:189–203, 2007.

R. Tempo, E.-W. Bai, and F. Dabbene. Probabilistic robustness analysis: Explicit bounds for the minimum number of samples. *Sys. Cont. Lett.*, 30:237–242, 1997.

R. Tempo, G. Calafiore, and F. Dabbene. *Randomized Algorithms for Analysis and Control of Uncertain Systems.* Springer-Verlag, London, 2005.

A. Tremba, G. Calafiore, F. Dabbene, E. Gryazina, B.T. Polyak, P. Shcherbakov, and R. Tempo. RACT – Randomized Algorithms Control Toolbox – User's Manual. 2007. URL `http://ract.sourceforge.net`.