User Guide for Simulation-Aided Robust Region-of-Attraction Analysis Software

1 Initializing the parameters used in the analysis and `GetRoaOpt.m`

`GetRoaOpt.m` creates/initializes the options and the system used in the analysis.

Syntax:

$$[\texttt{roaconstr}, \texttt{opt}, \texttt{sys}] = \texttt{GetRoaOpts}(\texttt{fWithDel}, \texttt{x}, \texttt{zV}, \texttt{p}, \texttt{Bis})$$

Inputs:

- fWithDel: the vector field as a vector of polynomials (contains both state variables and uncertain parameters)
- x: the state vector as a vector of fist degree monomials

The following inputs are optional and can be changed after calling `GetRoaOpt.m`. Pre-specifying them may be useful as the choices made on these parameters affect some of other parameters and GetRoaOpt.m initializes these extra parameters based on the pre-specified values.

- zV : [optional, default: purely quadratic monomials in x] basis vector for V
- p: [optional, default: $x^T x$] shape factor
- Bis: [optional] struct with the following fields.
  - Bis.flag: equal to 1 for SOS conditions with bisection, 0 for SOS conditions without bisection.
  - Bis.s2deg : [default: 1] HALF of the degree of s_2 (used when flag == 1)
  - Bis.r1deg : [default: 2] degree of r_1 (used when flag == 0)
  - Bis.r2deg : [default: 2] degree of r_2 (used when flag == 0)

Outputs: Three structs `roaconstr, opt,` and `sys`. Roughly, `roaconstr` holds the parameters used to specify the SOS problems (such as basis for V and the multipliers, l_1 and l_2, etc). `opt` holds the options for the code. `sys` is the structure describing the system in different forms, state vector, uncertain parameters (if any), etc. For a given problem, some of the fields of `sys` may not be used.

roaconstr.

zV	[monomials(x,2)] Basis for V.
p	[$x^T x$] Shape factor.
mu	[0] μ in $\dot{V} < -\mu V$
L1	[1e-6$x^T x$] l_1 in $V - l_1 \in \Sigma[x]$
L2	[1e-6$x^T x$] l_2 in $\dot{V}$ constraint
L2LP	[$x^T x$] This was experimental purposes - not used in any of the reported results. If `opt.findVfeas.feasonly == 0`, this has no effect.
z1	[chosen to match degrees in SOS condition based on p and zV] Basis for s_1 ($s_1 = z1'Qz1$)
z2	[monomials(x,1)] Basis for s_2 ($s_2 = z2'Qz2$)
z3	[chosen to match degrees in SOS condition based on z2] Basis for s_3 ($s_3 = z3'Qz3$)
zr1	[monomials(x,2)] Basis for r_1 ($r_1 = q'zr1$)
zr1	[monomials(x,2)] Basis for r_2 ($r_2 = q'zr2$)

opt.

opt.coordoptim.

flag	[0] equal to 1 for SOS conditions with bisection and 0 for SOS conditions without bisection
MaxIters	[20] Max number of iterations in coordinate-wise affine search
IterStopTol	[0.03] Stopping tolerance in coordinate-wise affine search
solver	['sedumi'] Name of the solver 'sedumi' or 'sdpt3' (This may not be used currently, meaning that everything is hardwired to 'sedumi')

opt.display.

sedumi	[0] turn SeDuMi display on and off, 0 for off and 1 for on
roaest	[0] turn all ROA estimation printouts on and off, 0 for off and 1 for on
BB	[1] turn summary outputs at the end of each B&B iterations on and off, 0 for off and 1 for on

opt.SaveResults.

filename	[''] Only used in B&B to save results on the go if a path to

a file is specified.

---

opt.parallel.

useCluster	[0] 1 to use cluster and 0 for not to use the cluster
machinesToUse	[1:9] double array with the ”node” ids to be used

---

opt.sim.

BetaInit	[5] Beta to run the first simulation
BetaShrink	[0.95] Factor by which Beta is decreased if there is a divergent trajectory or the LP is not feasible
BetaMax	[5] Max value of Beta for simulations
BetaGrow	[2] Factor by which Beta is increased
NumConvTraj	$[100 + 4^n]$ Number of convergent trajectories per system
tfinal	[100] Final time for simulations
tfinalBackForDiv	[10] Time to integrate a divergent trajectory backward in time
dispEveryNth	[100] display results during simulation after every dispEveryNth-th convergent simulation.
event_params	parameters for the event function to stop the simulations.

-.

opt.sim.event_params.

nbig	Stop simulation if $\|x(t)\| \geq nbig * \|x_0\|$ and assign divergent
nsmall	Stop simulation if $\|x(t)\| \leq nbig * \|x_0\|$ and assign convergent
xbig	Stop simulation if $\|x(t)\| \geq xbig$ and assign divergent
xsmall	Stop simulation if $\|x(t)\| \leq xsmall$ and assign convergent

-.

---

opt.lp.

useVdot	[1] If 1 use $\dot{V} < 0$ in forming LP, if 0 do not
useVdecr	[0] If 1 use $V(k+1) - V(k) < 0$ in forming LP, if 0 do not
decimN	[10] Take every decimN-th point on simulation trajectories
timestosolvelp	[40] Number of times to decrease Beta and resolve LP if the previous LP is infeasible
MinBetaToTryLP	[1e-3] Min value of Beta to try to solve the LP
sampleForPositive	[0] 1 if $V(x) \geq l_1(x)$ is imposed on sample point instead of imposing $V - l_1 \in \Sigma[x]$ (Experimental - not really used)
timesSampleLMI	[1000] Number of sample in $\{x : \|x\| \leq radiusSampleLMI\}$ when sampleForPositive = 1 (Experimental - not really used)
radiusSampleLMI	[0.1] radius of the ball to be sampled when sampleForPositive = 1 (Experimental - not really used)

---

opt.findVfeas.

usesosp	[0] 1 to use Pete's mysos. (Pete's mysos does not work.)
feasonly	[1] 1 to solve only a feasibility "LP" (This is only for experimental purposes, not used currently)
useYalmip	[1] 1 to solve the "LP" using Yalmip, 0 to use Pete's mysos.
sedopts.	To set SeDuMi options when Pete's mysos is used (Pete's mysos does not work.)

-.

opt.findVfeas.sedopts.

eps	[1e-9] Stopping tolerance for SeDuMi
fid	[opt.display.sedumi] 1 to display SeDuMi data
solver	['sedumi'] Name of the solver 'sedumi' or 'sdpt3' (This may not be used at the time being meaning that everything is hardwired to 'sedumi')
form	['image'] An option used in Pete's mysos when passing from the SOS problem to SDP.

-.

---

opt.sampleV.

nofVs	[5] Number of V's to be drawn from the convex outer bound for the original feasible set
nofptsEachTime	[5] Number of V's to be generated from which a sinlge V will be drawn as a Lyapunov function candidate (i.e., every nofptsEachTime-th point is chosen as a Lyapunov function candidate - to promote randomness in generating candidate V's
useLin	[1] 1 to use linearization constraints when sampling for V, 0 otherwise
useVsos	[1] 1 to use $V - l_1 \in \Sigma[x]$ constraint when sampling for V, 0 otherwise
Factor	[0.5] Stop sampling when sample V certifies $Factor * BetaSIM$ (when runtoNmax = 0)
runtoNmax	[0] 0 to stop sampling once $BetaCertified \geq Factor * BetaSIM$, 1 to generate nofVs many candidates and taking the best when if $BetaCertified \geq Factor * BetaSIM$ is satisfied

---

opt.getgamma.

bistol	[5e-3] Bisection stopping tolerance for gamma
maxgamma	[10] Only used for bisection
maxgammaBase	[opt.getgamma.maxgamma] opt.getgamma.maxgamma is changed at several places in the code. This is used to re-set it to its initial value
mingamma	[0] Only used for bisection
display	[opt.display.roaest] 1 to display getgamma printouts
usesosp	[0] 1 to use Pete's mysos (Pete's my sos does not work)
sedopts	[sedopts] See sedopts

---

opt.getbeta.

bistol	[5e-2] Bisection stopping tolerance for beta
maxbeta	[20] Only used for bisection
minbeta	[0] Only used for bisection
display	[opt.display.roaest] 1 to display getbeta printouts
usesosp	[0] 1 to use Pete's mysos (Pete's my sos does not work)
sedopts	[sedopts] See sedopts

---

opt.cellBeta.

NumberfsInFirstPhase	[0] 0 for center → vertices analysis, -1 for direct analysis with all vertices, positive integer for the number of systems to be sampled in Δ
checkPriorV	[0] If 0 or prior V is zero (or not passed in), run sim-based analysis directly. Else, assess the prior V first.

---

opt.BB.

max_iter	[20] Max number of $B\&B$ iterations
SubdivisionRule	['SubdivisionRule'] Name of the .m file which outputs the division rule
FactorNomSimCellBeta	[0.5]
FactorNomStartPriorCellBeta	[0.75]
NumberExtraNominal	[0] Number of extra vertex vector fields in addition to the center in cellBetaCenter
runBB	[0] 1 if wrapper is to run $B\&B$, 0 if wrapper is to run only cellBetaCenter
logToDiv	[zeros(length(sys.delvector))] i-th entry is 1 if the i-th direction is to be divided in log (base 10) scale.

---

sys.

---

delvector	vector of uncertain parameters (polys)
fWithDel	nx1 poly array, vector field with uncertain parameters
x	state vector (poly)
nonaffine	non-affine functions of uncertain parameters (polys)
f	nx(m+mpu+1) matrix of poly, fWithDel decomposed so that $fWithDel = f(:,1) + \sum_{i=1}^{m} f(:,i+1) * delvector(i) + \sum_{j=1}^{m_{pu}} f(:,j+m+1) * nonaffine(j)$

Example: For x the state vector, δ_1 and δ_2 uncertain parameters,

$$f(x,\delta) = f_0(x) + \delta_1 f_1(x) + \delta_2 f_2(x) + \delta_1^2 f_3(x)$$

sys is

$$sys.delvector = \delta$$
$$sys.fWithdel = f_0(x) + \delta_1 f_1(x) + \delta_2 f_2(x) + \delta_1^2 f_3(x)$$
$$sys.x = x$$
$$sys.nonaffine = [\delta_1^2]$$
$$sys.f = [f_0(x) \quad f_1(x) \quad f_2(x) \quad f_3(x)].$$

2 wrapper.m

This routine is for relatively easier use of several functions in in the analysis.

Syntax:

$$\texttt{outputs} = \texttt{wrapper}(\texttt{sys}, \texttt{c}, \texttt{roaconstr}, \texttt{opt})$$

See the help for `wrapper.m` for more detailed explanations. Roughly, it currently does one of the following depending on the form of the input:

- Given (non-uncertain) vector field compute the ROA
- Given uncertain vector field and a point in Delta, compute the ROA for the system corresponding to that point
- Given uncertain vector field and Delta, compute the robust ROA (center → vertices).
- Given uncertain vector field and Delta, run the branch-and-bound refinement

3 cellBetaCenter.m