

```

sdpvar v a t      %% 变量声明
sdpvar myeps0 myeps1 myeps2 myeps3

sf_low=-2.05;  %% 常量声明, 安全性质下、上边界
sf_upp=-1.95;

lsp=2500;      %% 常量声明, impulse

lambda1=.5;    %% 常量声明
lambda2=1;

tol0=1e-3;
tol1=1e-4;
tol2=1e-4;
tol3=1e-4;

[inv, c0] = polynomial([v a t], 6);  %% 模板多项式 inv 声明, 三变元六次多项式, 系数向量为
                                     %% c0, inv <= 0 为待求不变式模板

[s1, c1] = polynomial([v a t], 4);  %% SOS 形式多项式 s1,s2,s3,s4 声明, 待定系数向量分别为 c1,
                                     %% c2,c3,c4

[s2, c2] = polynomial([v a t], 4);
[s3, c3] = polynomial([v a t], 4);
[s4, c4] = polynomial([v a t], 4);

flow=[a-1.622; (1/lsp)*a^2; 1];  %% 微分方程声明
dinv=[jacobian(inv,v), jacobian(inv,a), jacobian(inv,t)];  %% 不变式多项式 inv 求偏导
lie=dinv*flow;  %% Lie 导数计算

%%-----

%% 初始条件约束
p_init1=replace(inv,v,-2);  %% 在 inv 中令 v=-2, a=1.622, t=0
p_init2=replace(p_init1,a,1.622);
p_init=replace(p_init2,t,0);

f0=p_init <= -myeps0;  %% 表示初始状态点满足 inv<=0
                       %% 考虑到数值计算误差, 可保留一定裕量 myeps0; 为简单计, 可
                       %% 将 myeps0 设为 0

%% Lie 导数约束
%% 要求 domain ==> -lie-lambda1*inv >= 0, lambda1 为可调节正常数, 经验取值区间为(0,1)

```

```

%% 可简单地要求 domain ==> lie < 0, 但采取上式求解不变式成功率更高
%% 此例中 domain 为  $t \geq 0 \wedge t \leq 0.128$ 

f1=sos(-lie-lambda1*inv - s1*t*(-t+0.128) - myeps1); %% Lie 导数约束的 sos 翻译, myeps1 为
%% 裕量

%% 安全性约束
%% domain  $\wedge$  not(safe) ==> inv > 0
%% 此例中 domain 为  $t \geq 0 \wedge t \leq 0.128$ , safe 为安全性质:  $v \geq sf\_low \wedge v \leq sf\_upp$ 
f2=sos(inv - s2*t*(-t+0.128) - s3*(v-sf_upp)*(v-sf_low) - myeps2); %% 安全性约束 sos 翻译,
%% 注意此处 myeps2 要求严格为正, 可调节大小

%% 归纳性约束
%% 要求跳转后仍位于不变式内, 即  $t=0.128 \implies inv(v',a',t') \leq 0$ , 其中  $t=0.128$  为 guard
%%  $v', a', t'$  由 reset 函数决定
%% 此例中采用条件:  $t=0.128 \implies inv(v,a',t') \leq \lambda_2 * inv(v,a,t)$ , 因其求解成功率更高
%% 其中 lambda2 为一正常数, 大小可调节, 一般取 1 即可

rst_inv1 = replace(inv, a, -0.01*(a-1.622)-0.6*(v+2)+1.622); %% reset 函数翻译
rst_inv = replace(rst_inv1, t, 0);

f3=sos(lambda2*inv - rst_inv - s4*(t-0.128) - myeps3); %% 归纳性约束翻译, myeps3
%% 为可调节正常数裕量, 可简单设为 0

%% -----
FeasibilityConstraints=[f0,f1,f2,f3,sos(s1),sos(s2),sos(s3),myeps0>=tol0,myeps1>=tol1,myeps2>=t
ol2,myeps3>=tol3]; %% 待求解约束集合, 其中 sos(s1)表示要求 s1 为一 SOS 形式的多项式,
%% sos(s2),sos(s3)为同一含义; 最后四个约束表示此处要求所有裕量均
%% 为正数, 其下界由设定的常数 tol1,tol2,tol3,tol0 决定

options=sdpsettings('verbose',1,'solver','sdpt3'); %% yalmip 求解选项设定
%% verbose 为 1 输出更多信息
%% 此处求解器 solver 选为 SDPT3
%options=sdpsettings('verbose',1,'solver','sedumi'); %% 或者可选为 Sedumi

%% 还可以输入其他一些参数给选定的求解器, 但一般选择默认参数, 即可省略此步
%options=sdpsettings('verbose',1,'solver','sedumi','sedumi.eps',1e-9); %% 一般省略
%options=sdpsettings('verbose',1,'solver','sdpt3','sdpt3.gaptol',1.0000e-009); %% 一般省略

solvesos(FeasibilityConstraints,[],options,[c0;c1;c2;c3;c4;myeps0;myeps1;myeps2;myeps3])
%% 调用 yalmip 求解命令 solvesos,
%% 第一个参数为所求约束, 第二个参数为优化目标, 此处为空
%% 第三个参数为选项, 第四个参数列出所有待定未知数

```

%% 若求解成功，会返回类似下面的信息

%ans =

%

% yalmiptime: 1.8230

% solvertime: 0.8220

% info: 'Successfully solved (SDPT3-4)'

% problem: 0

%% 打印所得不变式

mono\_inv=monolist([v a t], 6); %% 生成三变元六次单项式向量

pinv=double(c0')\*mono\_inv; %% double(c0): 获取 c0 向量的求解值，pinv<=0 为求得不变式

sdisplay(clean(pinv,1e-3)) %% 打印 pinv，保留小数点后三位