Case study: actuator (manipulated variables) and controlled variable selection, polyhedra vs SVD tools

Presentations in video:

http://personales.upv.es/asala/YT/V/sacerf1EN.html

http://personales.upv.es/asala/YT/V/sacerf2EN.html

http://personales.upv.es/asala/YT/V/sacerf3EN.html

http://personales.upv.es/asala/YT/V/sacerf4EN.html

This code runs in Matlab R2023a (Linux)

Objectives: understand SVD and polyhedron geometry and manipulations to asses wheter given setpoint increments or worst-case disturbance rejection are feasible without saturation.

Table of Contents

Model and constraints

Consider a linearised model $y_{2\times 1} = G_{2\times 3}u_{3\times 1} + H_{2\times 2}d_{2\times 1}$ with an operating point given by:

```
y_eq=[9 1.45]; u_eq=[2 0.5 1.7]; d_eq=[3 2];
```

where G and H are the transfer function matrices:

```
s=tf('s');
G=[3/(s+2)  0.15/(2*s+1)    -9/(0.3*s+1);
    0.15/(0.5*s+1)  -0.2/(s+1)  1.9/(0.3*s+1)];
H=[1.8/(s+1)^2 2/(s+3); 2.4/(s+12) 0.6/(s+1)/(s+3)];
```

Later on, we'll see that we can achieve all that we are required to... Hence, if we wish to test "actuator selection", using just two of them, we may make some columns of G equal to zero and execute the code again:

```
G=G*diag([1 1 1]); %set to zero position of actuator to disable.
```

Of course, "elliminating" one actuator should be, in rigor, "deleting" the column, but that would change the size of matrices so code would give errors. Setting column to zero is a quick workaround.

Manipulated variables u have the following saturation limits:

```
lim_u_abs=[4 1.5 2; 0 0 1]; %1st row max, 2nd row min
```

In incremental units, the limits of manipulated variables are:

```
inc_u_admissible=lim_u_abs-u_eq
```

```
inc_u_admissible = 2×3
    2.0000    1.0000    0.3000
   -2.0000   -0.5000   -0.7000
```

A) Reference tracking in steady state without saturation, SVD geometry

Basic computations

We wish to be able to move the outputs (via setpoint changes to be tracked by controllers) in the ranges:

```
lim_y_abs=[11.6 1.6; 8.5 1.35]; %1st row max, 2nd row min
```

So, in incremental units, these desired increments are:

```
inc_y_desired=lim_y_abs-y_eq
```

```
inc_y_desired = 2×2
    2.6000    0.1500
   -0.5000   -0.1000
```

Static DC gain matrix is:

```
Gain=dcgain(G)
```

```
Gain = 2×3
    1.5000    0.1500   -9.0000
    0.1500   -0.2000    1.9000
```

Scaling step: $y = E_y \cdot y_{esc}, \quad u = E_u \cdot u_{esc}$

```
Ey=diag(max(abs(inc_y_desired))) %worst case is maximum desired output increment
```

```
Ey = 2×2
    2.6000         0
         0    0.1500
```

```
Eu=diag(min(abs(inc_u_admissible))) %worst case is minimum available input increment
```

```
Eu = 3×3
    2.0000         0         0
         0    0.5000         0
         0         0    0.3000
```

The scaled gain matrix is given by:

$$y_{esc} = E_y^{-1} y = E_y^{-1} G u = \underbrace{E_y^{-1} G E_u}_{scaled \ \ gain} \cdot u_{esc}$$

```
Gain_scaled=inv(Ey)*Gain*Eu %Scaling for unit increments desired/available.
```

```
Gain_scaled = 2×3
    1.1538    0.0288   -1.0385
    2.0000   -0.6667    3.8000
```

```
svd(Gain_scaled)
```

```
ans = 2×1
    4.3646
    1.4985
```

Minimum gain is almost 1.5, above 1: satisfactory. Condition number is:

```
cond(Gain_scaled)
```

```
ans = 2.9127
```

The value of around 3 is quite sensible, lower than 5.

Hence, the answer is YES: we can move the outputs to the required amplitudes (steady state) with available inputs, when considering the geometry of ellipses, i.e., achieving any y_{esc} such that $\|y_{esc}\| = 1$ with $\|u_{esc}\| \leq 1$, norm is the Euclidean norm.

In a "pen-and-paper plus basic calculator" examination for my M.Sc. students this would finish the required answer.

Further discussion

We will now examine singular vectors (principal directions) for further insight:

```
[U,S,V]=svd(Gain_scaled)
```

```
U = 2×2
   -0.0991    0.9951
    0.9951    0.0991
S = 2×3
    4.3646         0         0
         0    1.4985         0
V = 3×3
    0.4298    0.8985    0.0891
   -0.1526   -0.0249    0.9880
    0.8899   -0.4382    0.1264
```

Intuitively, the "hard" manoeuver is, roughly, increase 1 unit output 1 and increase 0.1 units output 2 (grosso modo, keep it where it was), being manipulated variable 1 the most critical.

Also, as minimum gain is above $\sqrt{2}$, then the "unit sphere in u" will be able to achieve all output manoeuvers in the sphere of radius 1.49 in y space, which of course will include the "unit square in y, vertices at ± 1"; thus, the validity in the polyhedron geometry can be asserted without any polyhedron-specific code. This would not occur if the minimum gain were lower than $\sqrt{\text{num. outputs}}$.

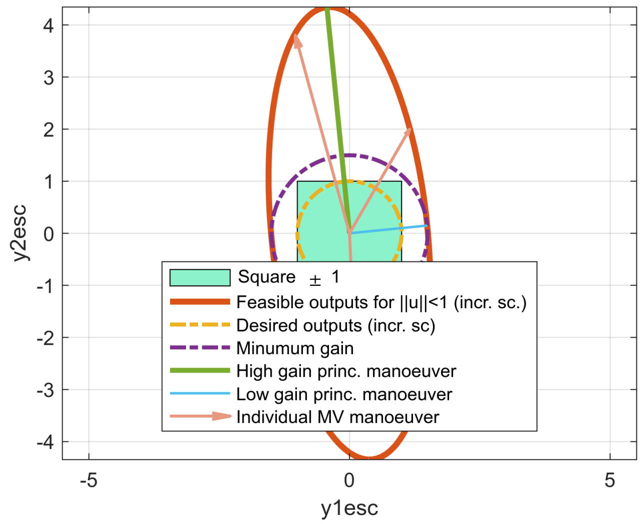
Let us graphically represent the above ideas.

If we draw the output ellipse swept by inputs $||u_{esc}|| = 1$ when multiplied by G_{esc}, we have:

```
M=Gain_scaled*Gain_scaled'
```

```
M = 2×2
    2.4106   -1.6577
   -1.6577   18.8844
```

```
syms y1 y2 real
y=[y1;y2];
fill([-1 -1 1 1],[-1 1 1 -1],[.55 0.95 .8]), hold on %unit square
fimplicit(y'*inv(M)*y-1, LineWidth=3), grid on %feasible ellipse
fimplicit(y'*y-1, '-.',LineWidth=2) %unit circle
fimplicit(y'*y-S(2,2)^2, '-.',LineWidth=2) %mingain circle
plot([0 U(1,1)*S(1,1)],[0 U(2,1)*S(1,1)],LineWidth=2.5) %output direction 1, scaled
plot([0 U(1,2)*S(2,2)],[0 U(2,2)*S(2,2)],LineWidth=1.25) %output direction 2, scaled
for i=1:3
    quiver(0,0,Gain_scaled(1,i),Gain_scaled(2,i),0,Color=[.9 .6 .5],LineWidth=1.5);
end
hold off, axis equal
xlabel("y1esc"),ylabel("y2esc")
legend("Square \pm 1","Feasible outputs for ||u||<1 (incr. sc.)","Desired outputs (inc
```



B) Steady state reference tracking, Polyhedron geometry

Basic computations

With polyhedron code, there is no need for "scaled" units, and in fact as ranges are quite asymmetric, even in scaled units the desired increments will not be "unity".

So, we'll work in original non-scaled units, but of course in INCREMENTAL ones, as we are working with a linear system. We have:

```
Vertices_incy=[8.5 8.5 11.6 11.6;1.35 1.6 1.6 1.35]-[9;1.45] %desired extreme points
```

```
Vertices_incy = 2×4
   -0.5000   -0.5000    2.6000    2.6000
   -0.1000    0.1500    0.1500   -0.1000
```

```
Vertices_incy=Vertices_incy; %scaling to check for extra margin
Min_incU=inc_u_admissible(2,:) %Lower Bound
```

```
Min_incU = 1×3
   -2.0000   -0.5000   -0.7000
```

```
Max_incU=inc_u_admissible(1,:) %Upper Bound
```

```
Max_incU = 1×3
    2.0000    1.0000    0.3000
```

We must check if there is a feasible u for each of the four extreme vertices of desired output.

The code below minimises $||u||^2$ subject to Gan·u=vertex , and subject to u being inside the maximum and minimum increment bounds.

```
quadprog(inv(Eu)^2,zeros(1,3),[],[],Gain,Vertices_incy(:,1),Min_incU,Max_incU) %factibl
```

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
ans = 3×1
   -0.4356
    0.0134
   -0.0168
```

```
quadprog(inv(Eu)^2,zeros(1,3),[],[],Gain,Vertices_incy(:,2),Min_incU,Max_incU) % NO fac
```

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
ans = 3×1
    0.0891
   -0.0170
    0.0701
```

```
quadprog(inv(Eu)^2,zeros(1,3),[],[],Gain,Vertices_incy(:,3),Min_incU,Max_incU) % NO fac
```

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
ans = 3×1
    1.4887
   -0.0248
   -0.0412
```

```
quadprog(inv(Eu)^2,zeros(1,3),[],[],Gain,Vertices_incy(:,4),Min_incU,Max_incU) %factibl
```

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
ans = 3×1
    0.9639
    0.0056
   -0.1281
```

As all vertices are feasible, we have proven that the requested steady state setpoint tracking problem is feasible without MV saturation, and we can achieve any point in the desired "output box". We knew it from the minimum-gain SVD computations, anyway.

Further discussion

Let us graphically represent the result of the above computations.

```
inc_u_admissible
```

```
inc_u_admissible = 2×3
    2.0000    1.0000    0.3000
   -2.0000   -0.5000   -0.7000
```

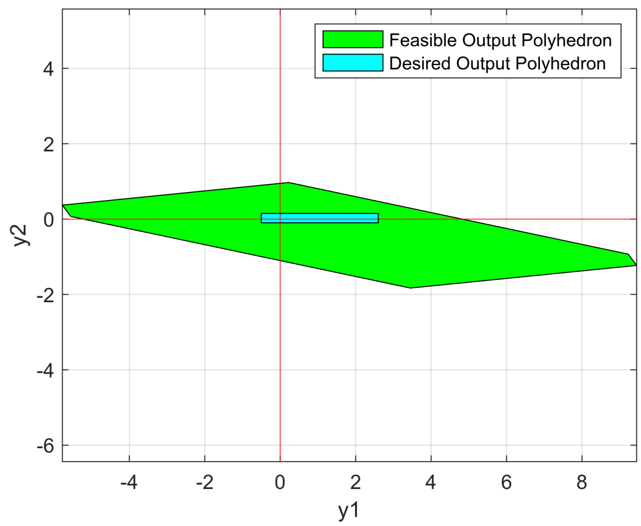
```
VerticesIncU=[2 1 .3;2 1 -.7;2 -.5 -.7;2 -.5 .3;-2 1 .3;-2 1 -.7;-2 -.5 -.7;-2 -.5 .3]'
```

```
VerticesIncU = 3×8
    2.0000    2.0000    2.0000    2.0000   -2.0000   -2.0000   -2.0000   -2.0000
    1.0000    1.0000   -0.5000   -0.5000    1.0000    1.0000   -0.5000   -0.5000
    0.3000   -0.7000   -0.7000    0.3000    0.3000   -0.7000   -0.7000    0.3000
```

```
ImageVerticesU=Gain*VerticesIncU
```

```
ImageVerticesU = 2×8
    0.4500    9.4500    9.2250    0.2250   -5.5500    3.4500    3.2250   -5.7750
    0.6700   -1.2300   -0.9300    0.9700    0.0700   -1.8300   -1.5300    0.3700
```

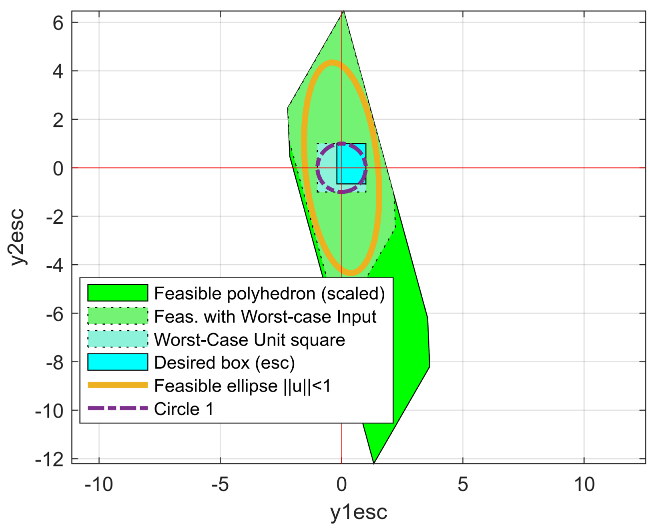
```
k=convhull(ImageVerticesU(1,:),ImageVerticesU(2,:)); %Order is important for "fill"
fill(ImageVerticesU(1,k),ImageVerticesU(2,k),'g') %feasible polyhedron
hold on
fill(Vertices_incy(1,:),Vertices_incy(2,:),'c')
hold off, grid on, axis equal
xlabel("y1"),ylabel("y2")
xline(0,'r'),yline(0,'r'), legend("Feasible Output Polyhedron","Desired Output Polyhedr
```



Even if it is not strictly needed, we can plot in "scaled" units, to compare with the ellipses, etc:

```
VerticesIncU_SquareWorstCase=[2 .5 .3;2 .5 -.3;2 -.5 -.3;2 -.5 .3;-2 .5 .3;-2 .5 -.3;-2
ImageVerticesUWC=Gain*VerticesIncU_SquareWorstCase;
fill(1/Ey(1,1)*ImageVerticesU(1,k),1/Ey(2,2)*ImageVerticesU(2,k),'g'),
hold on
fill(1/Ey(1,1)*ImageVerticesUWC(1,k),1/Ey(2,2)*ImageVerticesUWC(2,k),[0.45 0.95 0.45],I
fill([-1 -1 1 1],[-1 1 1 -1],[0.55 0.95 0.85],LineStyle=':'), hold on %unit square
```

```
fill(1/Ey(1,1)*Vertices_incy(1,:),1/Ey(2,2)*Vertices_incy(2,:),'c'),
fimplicit(y'*inv(M)*y-1, LineWidth=3), grid on %feasible ellipse
fimplicit(y'*y-1, '-.',LineWidth=2) %unit circle
hold off, axis equal, xline(0,'r'),yline(0,'r')
xlabel("y1esc"),ylabel("y2esc")
legend("Feasible polyhedron (scaled)","Feas. with Worst-case Input","Worst-Case Unit sc
```



On scaling and relation to predictive control or LQR cost index

If we wished to achieve a given setpoint minimising $||u||$ in "online" operation, we would end up doing something VERY similar to the above quadprog. MPC would use not just the DC gain, but the whole step response (DMC). In a general case, if each element of u has its own units and range, we should actually minimise $||u_{esc}||^2$ so that all increments are "comparable". We may switch to G_{esc} and scale output accordingly, or we may keep original units and minimise $||u_{esc}||^2 = ||E_u^{-1}u||^2$, from the fact that, by definition, $u = E_u u_{esc}$. This would just require to change the first argument to quadprog from "`eye(3)`" to "`inv(E_u)^2`".

C) Total disturbance rejection without saturation (steady state)

SVD geometry

With a model $y = Gu + Hd$, input to fully cancel the effect of disturbance is disturbance multiplied by the "feedforward gain matrix" $-G^{-1}H$ o; in the case G is not square, we need pseudoinverse (scaled):

```
interv_d=[3.8 2.4;2.4 1.35]; %range (absolute units) of the two disturbances.
operatingpoint_d=[3 2];
incr_d=interv_d-operatingpoint_d %incremental units
```

```
incr_d = 2×2
    0.8000    0.4000
```

```
   -0.6000   -0.6500
```

```
Ed=diag(max(abs(incr_d))) %worst case is maximum disturbance
```

```
Ed = 2×2
    0.8000         0
         0    0.6500
```

```
Hgesc=inv(Ey)*dcgain(H)*Ed;
%Actually, Ey is not relevant for total cancellation. Only Eu and Ed matter.
FeedForward=-pinv(Gain_scaled)*Hgesc
```

```
FeedForward = 3×2
   -0.4930   -0.2343
    0.0461    0.0338
   -0.0131   -0.0989
```

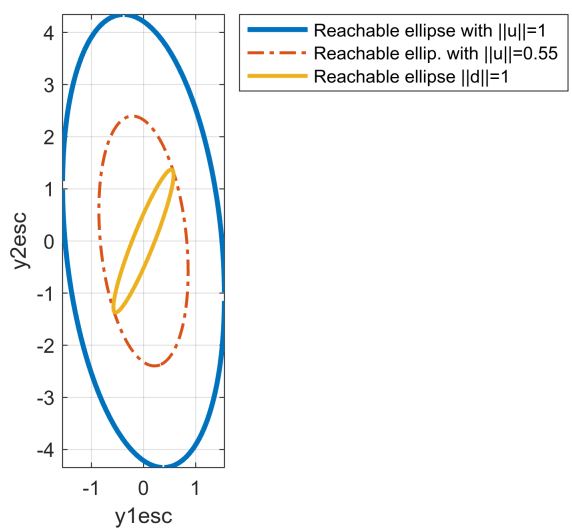
```
u_sv=svd(FeedForward)
```

```
u_sv = 2×1
    0.5515
    0.0838
```

Maximum gain <1 means that scaled disturbances $\|d_{esc}\| \leq 1$ can be cancelled with $\|u_{esc}\| \leq 0.552 < 1$.

Graphical representation

```
Md=Hgesc*Hgesc';
fimplicit(y'*inv(M)*y-1, LineWidth=2.5), grid on %feasible ellipse
hold on
%elipsoide que las entradas pueden mover
fimplicit(y'*inv(M)*y-u_sv(1)^2, '-.',LineWidth=1.5), grid on %feasible ellipse
fimplicit(y'*inv(Md)*y-1, LineWidth=2) %output ellipse in open loop due to disturbances
xlabel("y1esc"),ylabel("y2esc")
hold off, axis equal, legend("Reachable ellipse with ||u||=1","Reachable ellip. with ||
```



Indeed, we can see that the effect of u is larger than that of d, so u has no problems to counteract the "yellow" ellipsoid producing an output contrary to it inside the "red" ellipsoid.

Polyhedron geometry, total cancellation (steady state)

```
%zoomfactor=1;
zoomfactor=2.08 %for partial rejection, later on
```

```
zoomfactor = 2.0800
```

```
%zoomfactor=1.78 %max. feasible for total rejection below
Vertices_incD=[0.8 0.8 -0.6 -.6; 0.4 -0.65 0.4 -0.65]*zoomfactor
```

```
Vertices_incD = 2×4
    1.6640    1.6640   -1.2480   -1.2480
    0.8320   -1.3520    0.8320   -1.3520
```

```
1/zoomfactor
```

```
ans = 0.4808
```

*Minimising $||u_{scaled}||^2 = u^T(E_u^{-1})^2u$ would make pseudo-inverse results coincident with quadprog ones (if pseudo-inverse were feasible). Note, however, that actual "implementation" would need a feedforward (measurable disturbance) component, both here and in the pseudo-inverse SVD computations.

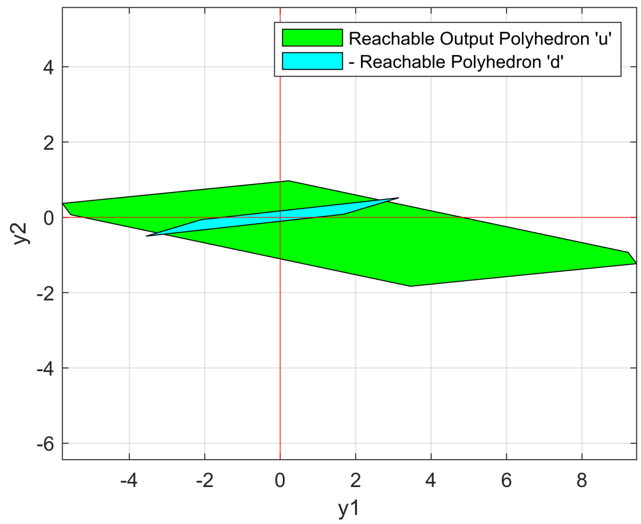
```
for i=1:4
    i
    quadprog(inv(Eu^2),zeros(1,3),[],[],Gain,-dcgain(H)*Vertices_incD(:,i),Min_incU,Max
end
```

```
i = 1
No feasible solution found.

quadprog stopped because it was unable to find a point that satisfies
the constraints within the value of the constraint tolerance.

<stopping criteria details>
i = 2
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
i = 3
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
i = 4
```

```
No feasible solution found.

quadprog stopped because it was unable to find a point that satisfies
the constraints within the value of the constraint tolerance.

<stopping criteria details>
```

So, we may plot the relevant polyhedra to check the meaning of the above. In Non-scaled coordinates:

```
ImageVerticesd=-dcgain(H)*Vertices_incD; %Change sign, because "u" must counteract the
kd=convhull(ImageVerticesd(1,:),ImageVerticesd(2,:)); %Order is important for "fill"


fill(ImageVerticesU(1,k),ImageVerticesU(2,k),'g') %feasible polyhedron
hold on
fill(ImageVerticesd(1,kd),ImageVerticesd(2,kd),'c')
hold off, grid on, axis equal
xlabel("y1"),ylabel("y2")
xline(0,'r'),yline(0,'r'), legend("Reachable Output Polyhedron 'u'","- Reachable Polyhe
```



Even if it is not strictly needed, we can plot in "scaled" output units, to compare with the ellipses, etc:

```
fill(1/Ey(1,1)*ImageVerticesU(1,k),1/Ey(2,2)*ImageVerticesU(2,k),'g'), %Reachable with
hold on
%reachable with worst-case U in scaling: cube of vertices +/-1.
fill(1/Ey(1,1)*ImageVerticesUWC(1,k),1/Ey(2,2)*ImageVerticesUWC(2,k),[0.45 0.95 0.45],I

%reachable with 'd' in worst-case unit scaled square
Vertices_incDSquare=[0.8 0.8 -0.8 -.8;0.65 -0.65 0.65 -0.65]; %worst-case "square" of c
ImageVerticesdSquare=-dcgain(H)*Vertices_incDSquare; %Change sign, because "u" must cou

fill(1/Ey(1,1)*ImageVerticesdSquare(1,kd),1/Ey(2,2)*ImageVerticesdSquare(2,kd),[0.45 0.
%reachable with non-symmetric rectangle in d
fill(1/Ey(1,1)*ImageVerticesd(1,kd),1/Ey(2,2)*ImageVerticesd(2,kd),'c'),
```

```
fimplicit(y'*inv(M)*y-1, LineWidth=2.5), grid on %reachable ellipse 'u'
fimplicit(y'*inv(Md)*y-1, LineWidth=2) %output ellipse in open loop due to disturbances
hold off, axis equal, xline(0,'r'),yline(0,'r')
xlabel("y1esc"),ylabel("y2esc")
legend("Reachable polyhedron 'u'","Reachable polyhedron with cube |u|<1","- Reachable p
```