# Van Der Pol Region of Attraction Problem

This demo will demonstrate how to use the ROA analysis tools. By increasing the degree of the Lyapunov function V(x) used to estimate the region of attraction, a larger ROA will be found.

## Contents


- Problem Statement:
- Setup Dynamics
- Quadratic V(x)
- Quartic V(x)
- Degree 6 V(x)


```
format('compact')
clear all
close all
```

## Problem Statement:

Given f, L1, L2, p, this code computes solutions to the problem maximize beta by choice of V, beta subject to:

$V(0) = 0$

$V \;\; L1 >= 0$

$\{x : p(x) \leq beta\} \subset \{x : V(x) \leq 1\} \subset \{x : Vdot(x) < 0\}$

where Vdot = jacobian(V,x)*f(x)

This code solves SOS relaxations for the above problem following the procedure:

1. Generate an initial V feasible for the above constraints. - use simulation data + linearization - use only linearization

2. Improve the estimate of the ROA by further optimization, namely iterating between optimizing over the choice of "multipliers" for given V and optimizing over the choice of V given the multipliers.

## Setup Dynamics

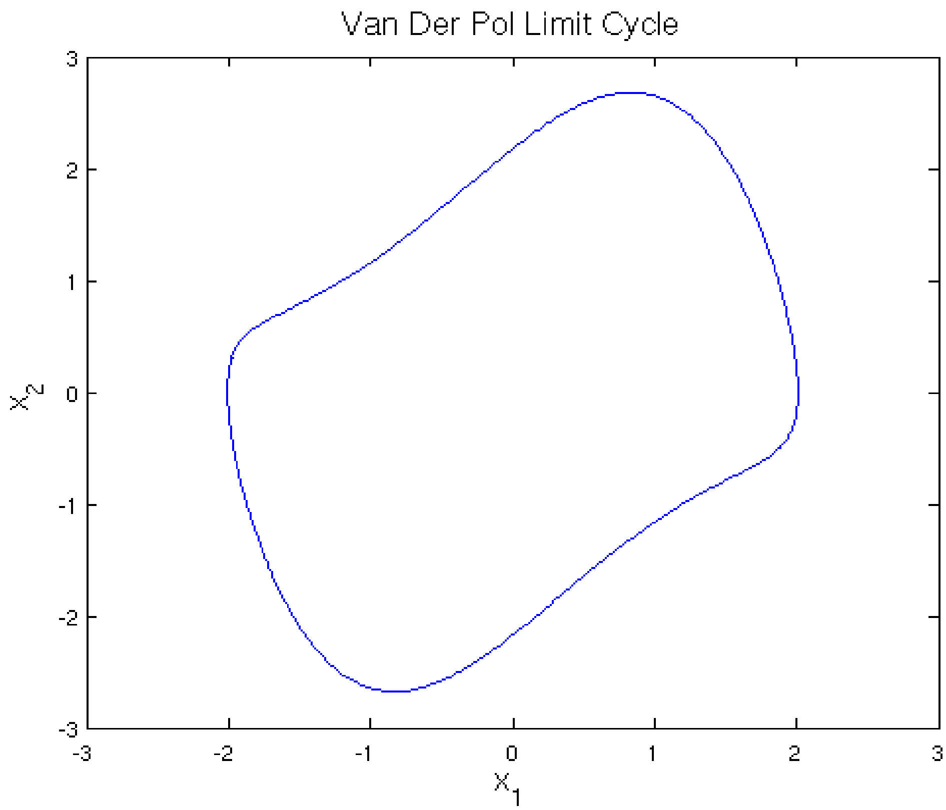
Form the vector field

```
pvar x1 x2;
x = [x1;x2];
```

```
x1dot = -x2;
x2dot = x1+(x1^2-1)*x2;
f= [x1dot; x2dot];
```

Plot Van Der Pol limit cycle

```
plotVDP
domain = [-3 3 -3 3];
axis(domain)
```




## Quadratic V(x)

**Extract default options.** The software uses two different iterations: one requires bisection and one does not. The default does not require bisection. The SOS multipliers in the two different iterations are defined as follows:

1. Without bisection: r1, r2 (these are polynomials in x - non necessarily SOS)

2. With bisection: s1, s2, s3 (all SOS)

Most of the slides are written based on the conditions that require bisection. Therefore, to ensure that we are using bisection we set

Bis.flag=1 (default = 0).

```
Bis.flag = 1;
```

**Generate the default options used for the rest of calculations.** We omit the 3rd and 4th input arguments for now.

```
[roaconstr,opt,sys] = GetRoaOpts(f, x, [],[],Bis);
```

**Set custom options.** We adjust some of the tolerance levels in order to get better results. By decreasing the iterations and increasing the tolerances, we get worse results, but the program runs faster.

```
opt.sim.NumConvTraj = 100;
opt.coordoptim.MaxIters = 30;
opt.coordoptim.IterStopTol = 1e-4;
opt.getbeta.bistol = 1e-4;
opt.getgamma.bistol = 1e-4;
opt.display.roaest = 1;
```

If Bis.flag = 1, then the default basis vectors for s1, s2, s3 are the following. (si is constructed as si = zi*Mi*zi with Mi SOS)

```
z1 = roaconstr.z1
z2 = roaconstr.z2
z3 = roaconstr.z3
```

```
z1 =
  1
z2 =
  [ x1 ]
  [ x2 ]
z3 =
  1
```

```
V is constructed as follows: V(x)= A*zV, where A is a row vector of
decision variables. The default vector field for V is quadratic.
```

```
zV = roaconstr.zV
```

```
zV =
  [  x1^2 ]
  [ x1*x2 ]
  [  x2^2 ]
```

We will use the unit ball as the default shape for p. p(x)=x'*x

```
roaconstr.p
```

```
ans =
  x1^2 + x2^2
```

We will use 1e-6*x'*x as the default shape for L1 and L2. L1 and L2 are small positive definite sums of squares.

```
roaconstr.L1
roaconstr.L2
```

```
ans =
  1e-06*x1^2 + 1e-06*x2^2
ans =
  1e-06*x1^2 + 1e-06*x2^2
```

**Estimate the ROA with Quadratic V(x).** The function wrapper estimates the ROA. The second input argument is empty because our vector field, f, does not have any uncertainty.

```
outputs = wrapper(sys,[],roaconstr,opt);
```

```
------------------Beginning simulations
System 1: Num Stable = 1      Num Unstable = 1       Beta for Sims = 2.348   Beta UB = 2.348
System 1: Num Stable = 52     Num Unstable = 2       Beta for Sims = 2.231   Beta UB = 2.347
System 1: Num Stable = 100    Num Unstable = 2       Beta for Sims = 2.231   Beta UB = 2.347
------------------End of simulations
------------------Begin search for feasible V
Try = 1        Beta for Vfeas = 2.231
Try = 2        Beta for Vfeas = 2.119
------------------Found feasible V
Initial V (from the cvx outer bnd) gives Beta = 1.495
-------------------Iteration = 1
Beta = 1.495 (Gamma = 0.739)
```

Get the V, betas and multipliers

```
[V,betaLower,gamma,p,multip,betaUpper] = extractSol(outputs);

V
s1 = multip.S1
s2 = multip.S2
betaLower
betaUpper
```

```
V =
```

```
  0.42647*x1^2 - 0.19336*x1*x2 + 0.35769*x2^2
s1 =
  0.49469
s2 =
  0.91877*x1^2 - 0.27698*x1*x2 + 2.4829*x2^2
betaLower =
    1.4947
betaUpper =
    2.2306
```

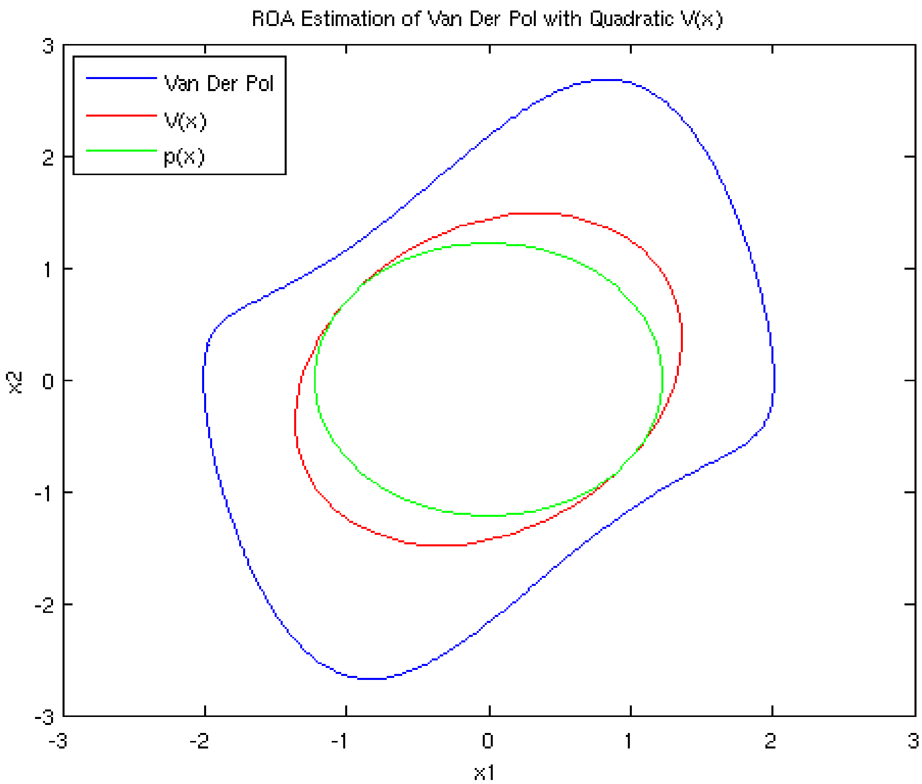
## Verify polynomials are SOS

```
issos(V)
issos(s1)
issos(s2)
```

```
ans =
     1
ans =
     1
ans =
     1
```

## **Display Results.** Plot p(x) and Quadratic V(x)

```
hold on;
betaLower=double(betaLower);
pcontour(V,gamma,domain,'r')
pcontour(p,betaLower, domain,'g')
title('ROA Estimation of Van Der Pol with Quadratic V(x)')
legend('Van Der Pol', 'V(x)', 'p(x)', 'Location', 'NorthWest')
axis(domain)
```




## Quartic V(x)

Get default options again. This time zV specifies that V is quartic.

```
zV = monomials(x, 2:4);
[roaconstr,opt,sys] = GetRoaOpts(f, x, zV,[],Bis);
```

A larger ROA is obtained by searching over quartic s2 polynomials

```
roaconstr.z2 = monomials(x, 1:2);
```

**Set custom options.** We need to specify the settings again.

```
opt.sim.NumConvTraj = 100;
opt.coordoptim.MaxIters = 30;
opt.coordoptim.IterStopTol = 1e-4;
opt.getbeta.bistol = 1e-4;
opt.getgamma.bistol = 1e-4;
opt.display.roaest = 0;
```

Estimate the ROA with Quartic V(x)

```
outputs = wrapper(sys,[],roaconstr,opt);
```

## Get the V, betas and multipliers

```
[V,betaLower,gamma,p,multip,betaUpper] = extractSol(outputs);

V
s1 = multip.S1
s2 = multip.S2
betaLower
betaUpper
```

```
V =
  0.0098431*x1^4 + 0.057867*x1^3*x2 + 0.070016*x1^2*x2^2
  - 0.045388*x1*x2^3 + 0.014722*x2^4 + 8.6629e-14*x1^3
  - 1.4808e-12*x1^2*x2 + 1.128e-13*x1*x2^2 + 1.1221e-12
  *x2^3 + 0.19913*x1^2 - 0.27147*x1*x2 + 0.14968*x2^2
s1 =
  0.13868*x1^2 + 0.099347*x1*x2 + 0.13018*x2^2 + 2.6481e
  -09*x1 + 2.2007e-09*x2 + 0.22947
s2 =
  0.90965*x1^4 - 0.54333*x1^3*x2 + 2.5467*x1^2*x2^2
  - 0.84735*x1*x2^3 + 0.14879*x2^4 + 2.3313e-11*x1^3
  - 2.5534e-11*x1^2*x2 - 5.8705e-11*x1*x2^2 + 2.8545e-11
  *x2^3 + 0.74107*x1^2 - 0.46906*x1*x2 + 0.076265*x2^2
betaLower =
    2.1413
betaUpper =
    2.3492
```

## Verify polynomials are SOS

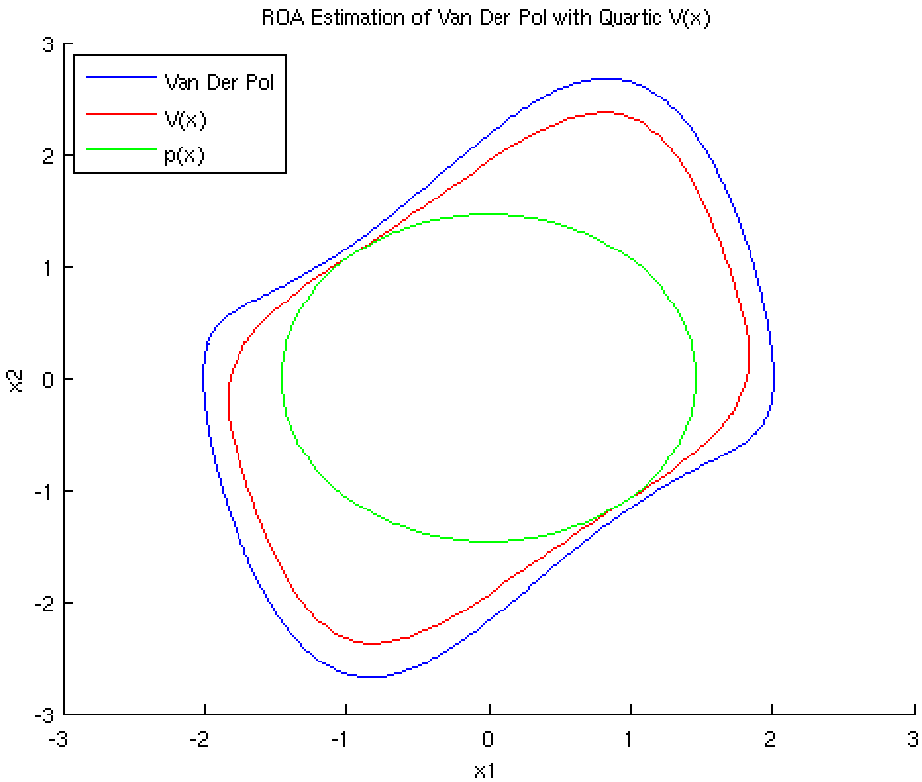
```
issos(V)
issos(s1)
issos(s2)
```

```
ans =
     1
ans =
     1
ans =
     1
```

## Plot p(x) and V(x)

```
figure;
hold on;
plotVDP
```

```
pcontour(V,gamma,domain,'r')
pcontour(p,betaLower, domain,'g')
title('ROA Estimation of Van Der Pol with Quartic V(x)')
legend('Van Der Pol', 'V(x)', 'p(x)', 'Location', 'NorthWest')
axis(domain)
```




## Degree 6 V(x)

Get default options again. This time zV specifies that the degree of V is 6.

```
zV = monomials(x, 2:6);
[roaconstr,opt,sys] = GetRoaOpts(f, x, zV,[],Bis);
```

A larger ROA is obtained by searching over quartic s2 polynomials

```
roaconstr.z2 = monomials(x, 1:2);
```

**Set custom options.** We need to specify the setting again.

```
opt.sim.NumConvTraj = 100;
opt.coordoptim.MaxIters = 30;
opt.coordoptim.IterStopTol = 1e-4;
```

```
opt.getbeta.bistol = 1e-4;
opt.getgamma.bistol = 1e-4;
opt.display.roaest = 0;
```

Estimate the ROA with degree 6 V(x) wrapper computes the ROA estimation routine. The second input argument is empty because our vector field, f, does not have any uncertainty. See "help wrapper" for instructions on specifying uncertain vector fields.

```
outputs = wrapper(sys,[],roaconstr,opt);
```

Get the V, betas and multipliers

```
[V,betaLower,gamma,p,multip,betaUpper] = extractSol(outputs);

V
s1 = multip.S1
s2 = multip.S2
betaLower
betaUpper


V =
  0.024602*x1^6 + 0.046349*x1^5*x2 - 0.0091987*x1^4*x2^2
  - 0.024579*x1^3*x2^3 + 0.041076*x1^2*x2^4 - 0.016574*x1
  *x2^5 + 0.0040183*x2^6 - 6.8829e-07*x1^5 + 2.2268e-06
  *x1^4*x2 - 4.454e-06*x1^3*x2^2 - 5.7385e-06*x1^2*x2^3
  + 3.6124e-06*x1*x2^4 - 2.5847e-06*x2^5 - 0.096009*x1^4
  - 0.11969*x1^3*x2 + 0.23584*x1^2*x2^2 - 0.12469*x1
  *x2^3 + 0.017174*x2^4 + 5.6344e-06*x1^3 + 1.6572e-06
  *x1^2*x2 + 4.3143e-06*x1*x2^2 + 1.3673e-05*x2^3
  + 0.44801*x1^2 - 0.27595*x1*x2 + 0.21466*x2^2
s1 =
  0.14772*x1^4 + 0.045747*x1^3*x2 + 0.11794*x1^2*x2^2
  + 0.039732*x1*x2^3 + 0.14143*x2^4 - 2.0472e-06*x1^3
  + 2.5874e-06*x1^2*x2 - 9.2161e-06*x1*x2^2 - 2.7463e-06
  *x2^3 + 0.031121*x1^2 + 0.12375*x1*x2 + 0.12099*x2^2
  + 7.3059e-06*x1 - 9.7849e-06*x2 + 0.60248
s2 =
  1.1735*x1^4 - 2.6837*x1^3*x2 + 2.4065*x1^2*x2^2
  - 0.022651*x1*x2^3 + 0.067581*x2^4 - 6.5573e-05*x1^3
  - 9.7436e-05*x1^2*x2 - 0.00024776*x1*x2^2 + 8.7312e-05
  *x2^3 + 0.15422*x1^2 - 0.023624*x1*x2 + 0.063628*x2^2
betaLower =
    2.3151
betaUpper =
    2.3478
```

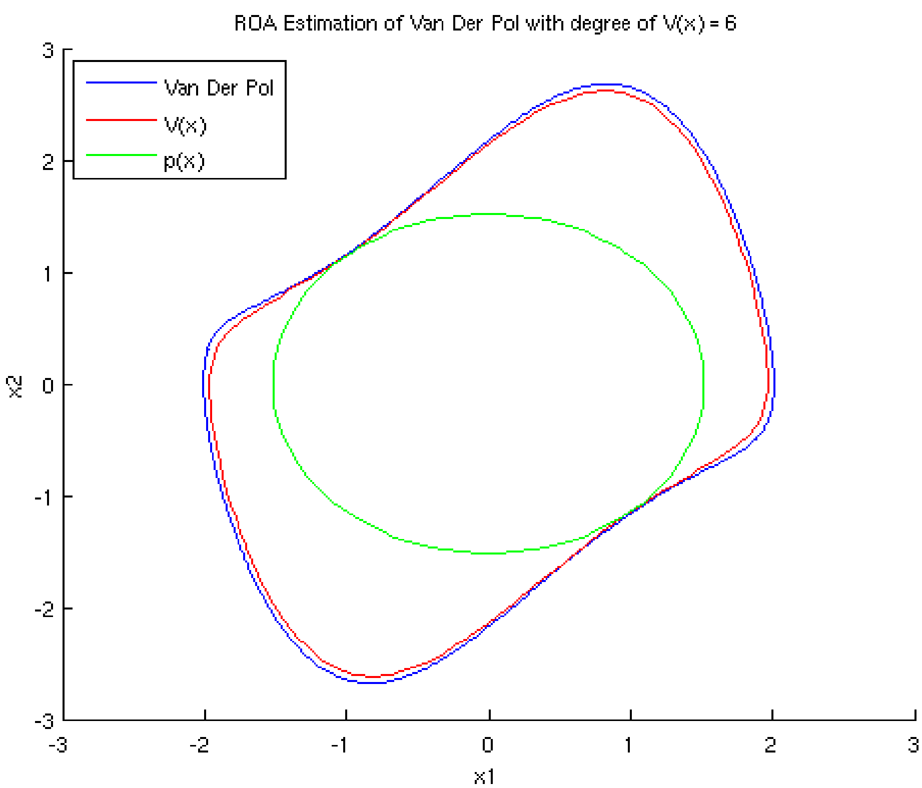
Verify polynomials are SOS

```
issos(V)
issos(s1)
issos(s2)


ans =
     1
ans =
     1
ans =
     1
```

Plot p(x) and degree 6 V(x)

```
figure;
hold on;
plotVDP
pcontour(V,gamma,domain,'r')
pcontour(p,betaLower, domain,'g')
title('ROA Estimation of Van Der Pol with degree of V(x) = 6')
legend('Van Der Pol', 'V(x)', 'p(x)', 'Location', 'NorthWest')
axis(domain)
```




For details of iterations with no bisection see:

*Topcu, Seiler, and Packard, "Local Stability Analysis Using Simulations and Sum-of-Squares Programming," Automatica, 2008 or the slide titled "Application of Set Containment Conditions (2)."*