Example: exercise 8

I express my way of using matlab with couple of examples.

You can find solutions to any problem by asking google "how to do this and that in matlab" or just searching matlab's help, it's pretty good. This is how I have learned everything I know about matlab. I might do some funny things as nobody has taught me, but at least this works :)

Non-bolf text after ">>" sign is command I gave to matlab in command window. Non-bold without ">>" is matlab's output in command window. Commands could be collected to a script to save and use later etc. but usually I try them at least once in command window to get immediate response whether syntax etc. are ok.

Create symbolic variables you need:

```
>> syms a q S I positive
>>
```

Adding "positive" at the end sets condition to those variables. It's not necessary.

Create symbolic function for parameter b:

```
>> b = symfun(3 + 2*a^2/(1+0.2*a^2), a)

b(a) =

(2*a^2)/(a^2/5 + 1) + 3

>>
```

If you want to avoid output use semicolon at the end:

```
>> b = symfun(3 + 2*a^2/(1+0.2*a^2), a);

>>
```

Syntax: "symfun(formula, inputs)". Another way of creating symfun:

```
>> b(a) = 3 + 2*a^2/(1+0.2*a^2)

b(a) =

(2*a^2)/(a^2/5 + 1) + 3

>>
```

For creating symfun of variable a you have to have defined symbolic variable a.

Now I define symbolic variables dS and dI that contain right-hand sides of our system. I could also define them as symbolic functions but I see no need for that.

```
>> dS = 2*S - (S+I)*S - S*I*b(a)

dS =
```

```
2*S - S*(I + S) - I*S*((2*a^2)/(a^2/5 + 1) + 3)

>> dI = S*I*b(a) - a*I - (S+I)*I;

>>
```

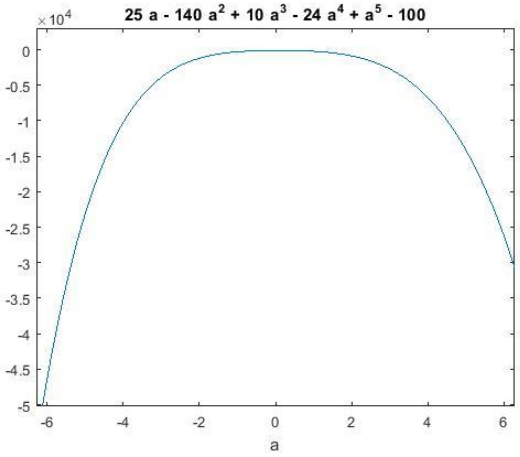
Now I solve equilibrium of the system i.e. S and I for resident a.

```
>> [Sres,Ires] = solve(dS == 0, dI == 0, [S, I])

Warning: The solutions are valid under the following conditions: 25*a - 140*a^2 + 10*a^3 - 24*a^4 + a^5 -
100 < 0. To include parameters

and conditions in the solution, specify the 'ReturnConditions' option.

> In solve>warnIfParams (line 517)

  In solve (line 367)

Sres =

(2*(7*a^5 + a^4 + 45*a^3 + 10*a^2 + 50*a + 25))/(169*a^4 + 390*a^2 + 225)

Ires =

-(a^5 - 24*a^4 + 10*a^3 - 140*a^2 + 25*a - 100)/(169*a^4 + 390*a^2 + 225)

>>
```

Matlab is not very good with symbolic calculations, it's more for numerics. Thus it sometimes gives this kind of conditions that it's not able to handle by itself. This time condition comes from the fact that I defined "I" to be positive. Formula in condition is in numerator of Ires. To check what does the condition mean I plot it:

```
>> ezplot(25*a - 140*a^2 + 10*a^3 - 24*a^4 + a^5 - 100)

>>
```

ezplot is an easy-to-use plotter for symbolic expressions. I strongly recommend to learn using those ez-functions. Read matlab help for that :) ezplot opens this figure:

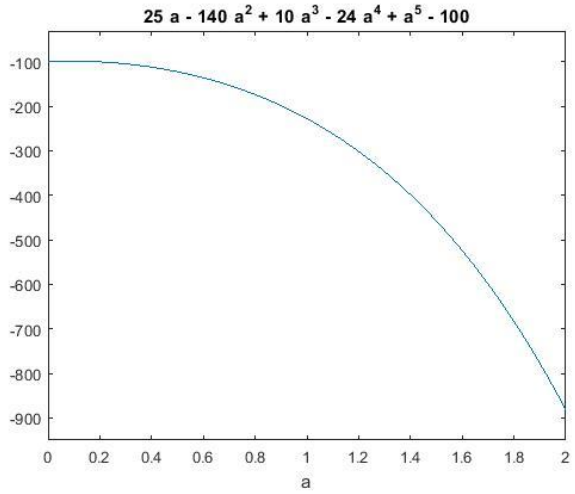


ezplot uses [-2pi,2pi] as default domain for a plot. As exercise suggests studying range 0 <= a <= 2 we can plot the condition on that range to make sure the formula is negative and thus I is positive:

>> ezplot(25*a - 140*a^2 + 10*a^3 - 24*a^4 + a^5 - 100, [0,2])

>>

[0,2] at the end defines the domain. No I got:



Clearly negative, good!

Now I define invasion fitness of mutant q as a symbolic variable s:

>> s = Sres*b(q)-q-(Sres + Ires)

s =

(a^5 - 24*a^4 + 10*a^3 - 140*a^2 + 25*a - 100)/(169*a^4 + 390*a^2 + 225) - (2*(7*a^5 + a^4 + 45*a^3 + 10*a^2 + 50*a + 25))/(169*a^4 + 390*a^2 + 225) - q + (2*((2*q^2)/(q^2/5 + 1) + 3)*(7*a^5 + a^4 + 45*a^3 + 10*a^2 + 50*a + 25))/(169*a^4 + 390*a^2 + 225)

>>

As I said, matlab is not ideal for symbolics. Thus it gives for example awfully long formulas and even if you try to simplify them it not always work. This has been problem for me when doing my master's thesis. Matlab is unable to simplify my functions and thus uses tens and hundreds of thousands characters for them, and gets too slow to do anything. But I asked Eva to simplify them with mathematica and got formulas of length 3000 or so. So better buy mathematica. But matlab is not totally dummy. It can still do something:

>> simplify(s)

ans =

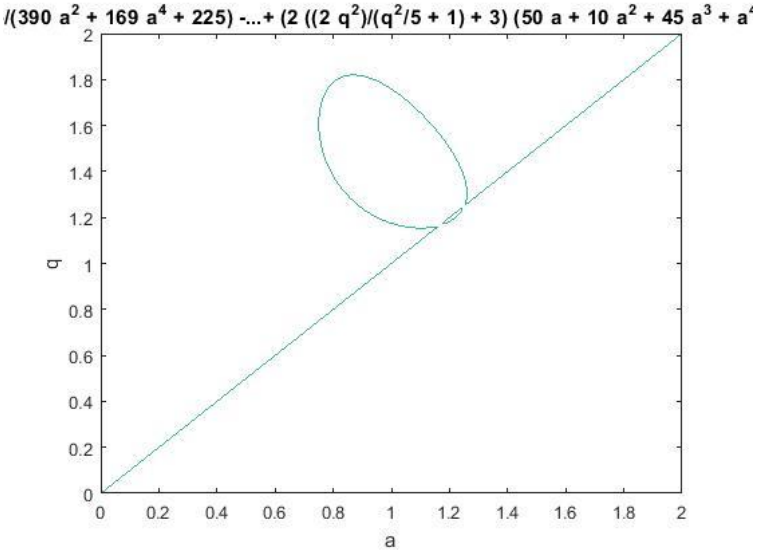
(169*a^5*q^2 + 145*a^5 - 169*a^4*q^3 - 845*a^4*q - 100*a^4 + 1090*a^3*q^2 + 950*a^3 - 390*a^2*q^3 + 100*a^2*q^2 - 1950*a^2*q - 500*a^2 + 1225*a*q^2 + 1125*a - 225*q^3 + 500*q^2 - 1125*q)/((q^2 + 5)*(13*a^2 + 15)^2)

>>

Bit shorter! :D I suggest studying this simplify –function from help. There is quite some possibilities to use it. Anyway, now I have invasion fitness, thus I draw a PIP:

>> ezplot(s==0, [0,2])

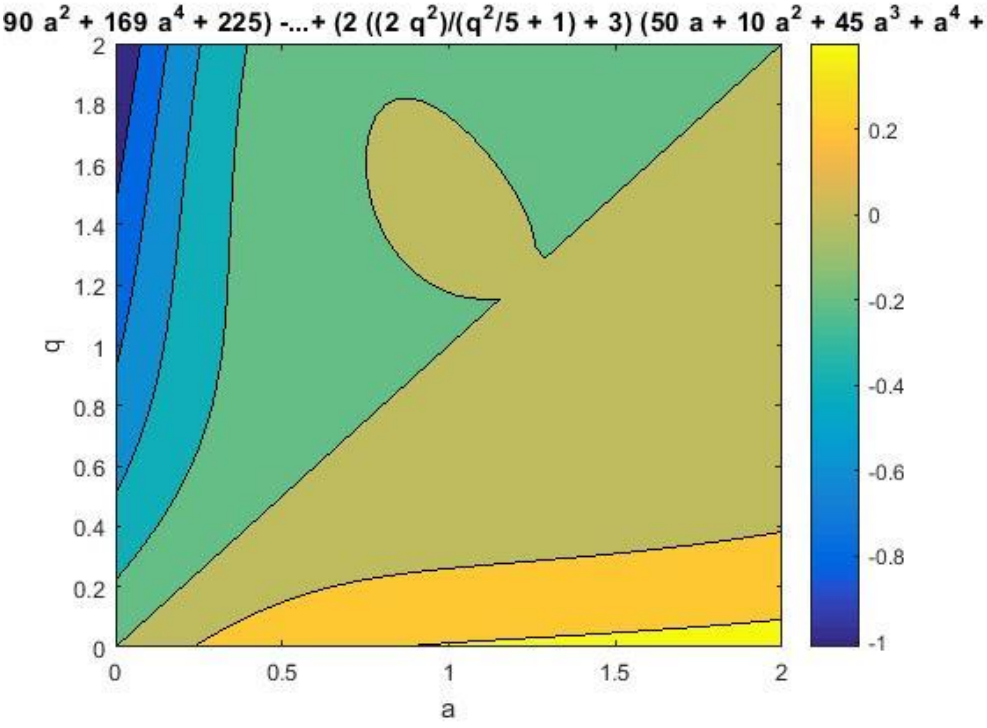
>>



This drew invasion boundary, but how to know where is positive and where negative? For example I could calculate value of s in one or two points and define sings that way or I can draw a contour plot and insert a colorbar to it in figure window:

>> ezcontourf(s, [0,2])

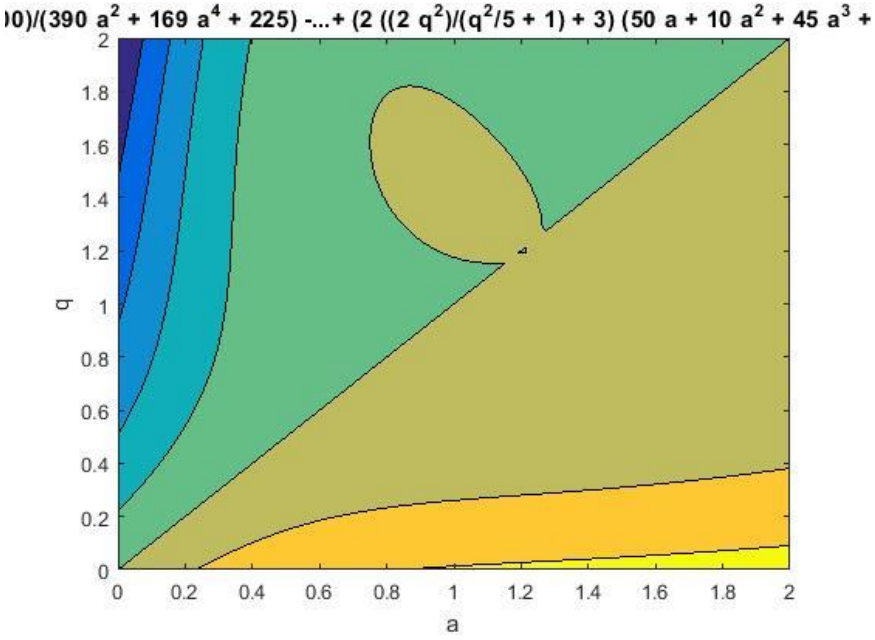
>>



Pretty nice! Bubble is positive as is lower part also. But close to diagonal the figure it's not very accurate. To make it better I can increase number of sample points. Default for ez-functions is 60, or 60x60 in this example. Let's try 100.

>> ezcontourf(s, [0,2], 100)

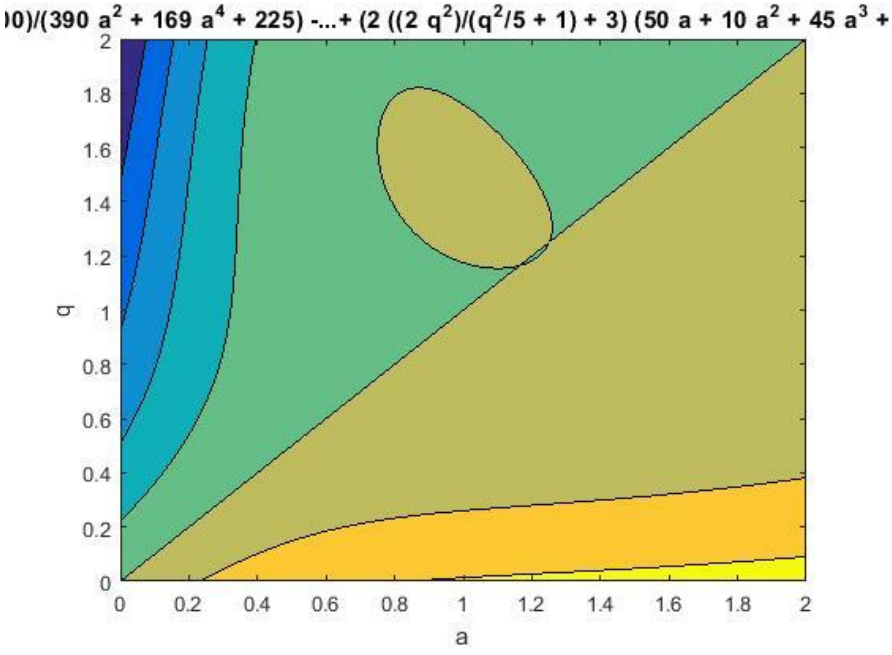
>>



Better but not enough.

>> ezcontourf(s, [0,2], 1000)

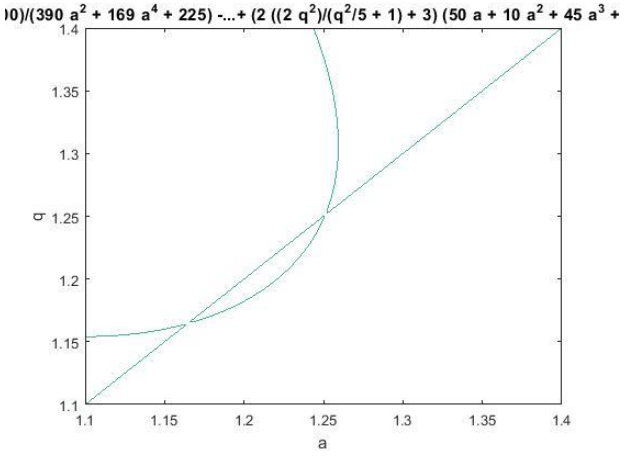
>>



Now it's good!

Now we want to solve those singularities and characterize them. Let's first get better sight to them:

```
>> ezplot(s, [1.1,1.4], 100)

>>
```



I will use this information for solving them more accurately. Matlab's numerical solver "vpasolve" accepts interval of search or initial conditions, again read help. I also use function "subs" to substitute symbolic variables inside a symbolic variable. But first I define selection gradient on diagonal:

```
>> grad = subs(diff(s, q), q, a)

grad =

(2*((4*a)/(a^2/5 + 1) - (4*a^3)/(5*(a^2/5 + 1)^2))*(7*a^5 + a^4 + 45*a^3 + 10*a^2 + 50*a + 25))/(169*a^4 + 390*a^2 + 225) - 1

>>
```

diff(function, variable) makes differentiation and subs(function, old_variable, new_variable) takes us to diagonal. Now singularities are roots of grad. We could try to solve them explicitly but it seem not to work, thus I use numerics:

```
>> vpasolve(grad==0)

ans =

1.164478253570660283274231929077

1.251083465393448822214031986082

0.184774892098195107617809817686O5 - 1.264518935395987104258444160268 3i

-1.392555751580249660361941775265 5 + 0.926619395057817361479903265394 69i

-1.392555751580249660361941775265 6 - 0.926619395057817361479903265394 67i

0.184774892098195107617809817686O6 + 1.264518935395987104258444160268 3i

>>
```

Above is pretty clear explanation why defining an interval is useful :D vpasolve ignores assumptions we have done on symbolic variables (e.g. a positive). We can make pretty sure that we get only on solution and then collect those values to own variables:

```
>> sing1 = vpasolve(grad==0, [1.15,1.2])

sing1 =

1.164478253570660283274231929077

>> sing2 = vpasolve(grad==0, [1.2,1.3])

sing2 =

1.251083465393448822214031986082

>>
```

Now we have numerical values of singularities. Then second derivatives:

```
>> double(subs(diff(s, q, 2), {q, a}, {sing1, sing1}))

ans =

   0.1259

>> double(subs(diff(s, a, a), {q, a}, {sing1, sing1}))

ans =

   0.0440

>>
```

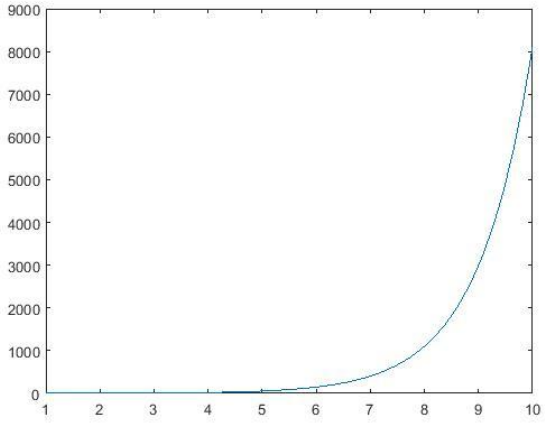
Thus at lower singularity $\partial_{yy}s > \partial_{xx}s > 0$. Eight cases -> invadable repelling. And same for the upper one:

```
>> double(subs(diff(s, q, q), {q, a}, {sing2, sing2}))

ans =

   0.0371

>> double(subs(diff(s, a, 2), {q, a}, {sing2, sing2}))

ans =

   0.1137

>>
```

Thus at upper singularity $0 < \partial_{yy}s < \partial_{xx}s$ => attracting invadable = BP.

Now shortly about canonical equation. "[T, Y] = ode45(fun_handle, [t0,t], y0)" solves differential equation of form dy/dt = f(t,y) where fun_handle contains the right-hand side i.e. f(t,y). Handle to function can be created with "@". Function can be anonymous: @(t,y)y creates handle to a function that accepts input arguments t and y and returns y. Read "anonymous functions" help. To solve dy/dt=y in the interval [1,10] and initial condition y(t0) = y(1) = 1 we do:

```
>> [T,Y] = ode45(@(t,y)y, [1,10], 1);

>> plot(T,Y)

>>
```



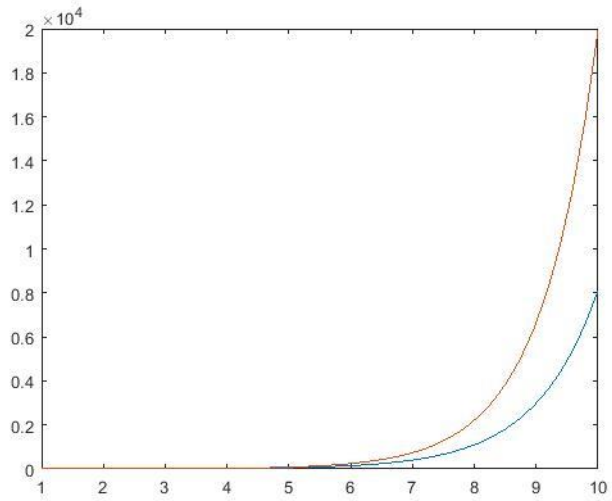
We can solve system of equations by defining column vector of right-hand sides. To solve dy(1)/dt = y(1) & dy(2)/dt = 1.1*y(2) we do:

```
[T,Y] = ode45(@(t,y)[y(1);1.1*y(2)], [1,10], [1,1]);

>> plot(T,Y)
```

Now output Y is n*2 matrix where first column contains values for y(1) and second for y(2). Plot works nicely with that directly. We could also plot them separately:

```
>> plot(T,Y(:,1))

>> hold on

>> plot(T,Y(:,2))

>>
```

Anyway, result is:

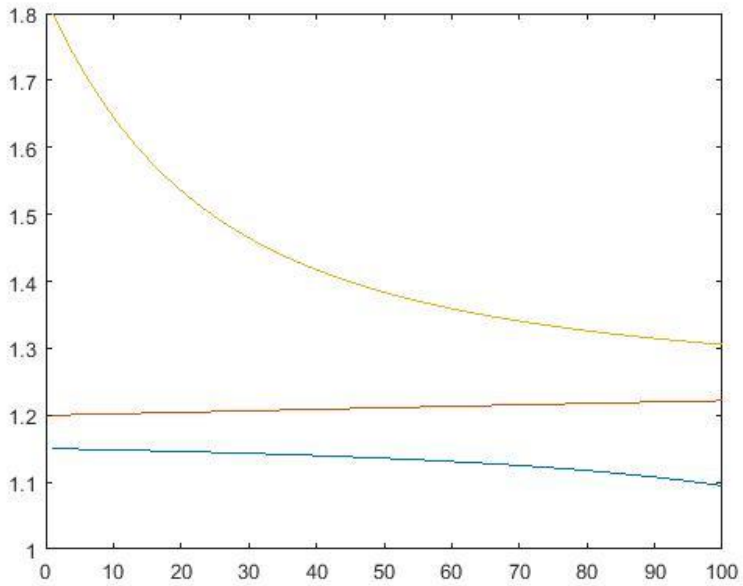


We can also define non-anonymous function and create handle to that. For canonical equation I made a function "canonical":

```
function [ can ] = canonical( t,a )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
Ires = -(a^5 - 24*a^4 + 10*a^3 - 140*a^2 + 25*a - 100)/(169*a^4 + 390*a^2 +
225);
grad = (2*((4*a)/(a^2/5 + 1) - (4*a^3)/(5*(a^2/5 + 1)^2))*(7*a^5 + a^4 + 45*a^3
+ 10*a^2 + 50*a + 25))/(169*a^4 + 390*a^2 + 225) - 1;
can = Ires*grad;
end
```

And saved it in file canonical.m. This is my caricature of canonical equation, I took only resident population density Ires and selection gradient grad that I solved earlier. This I can use with ode45 like that:

```
>> [T,A] = ode45(@canonical, [1,100], 1.15);
>> plot(T,A)
>> hold on
>> [T,A] = ode45(@canonical, [1,100], 1.2);
>> plot(T,A)
>> [T,A] = ode45(@canonical, [1,100], 1.8);
>> plot(T,A)
>>
```



Another possibility to do these three cases is to make immediately function file that returns for example three independent monomorphic canonical equation:

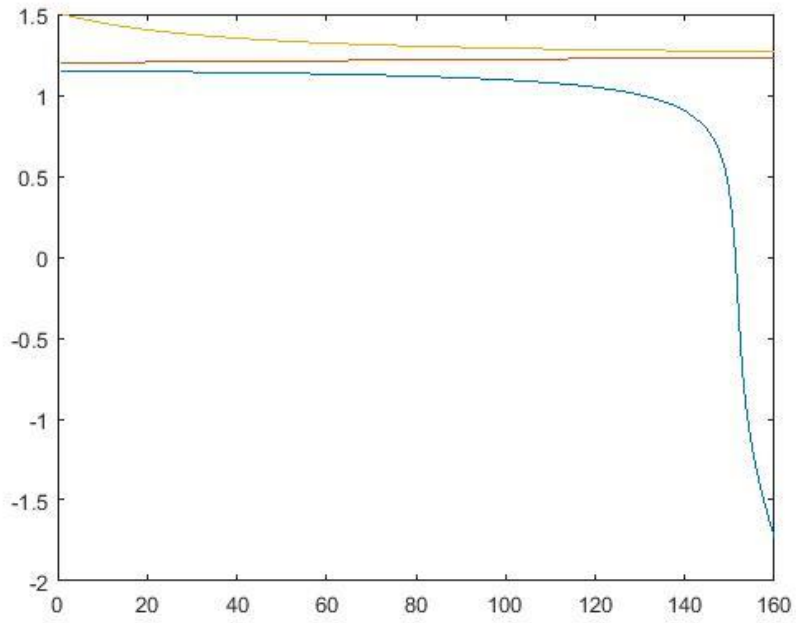
```
function [ can ] = canonical3( t,a )
%UNTITLED6 Summary of this function goes here
%   Detailed explanation goes here
can = zeros(3,1);
syms x
Ires(x) = -(x^5 - 24*x^4 + 10*x^3 - 140*x^2 + 25*x - 100)/(169*x^4 + 390*x^2 +
225);
grad(x) = (2*((4*x)/(x^2/5 + 1) - (4*x^3)/(5*(x^2/5 + 1)^2))*(7*x^5 + x^4 +
45*x^3 + 10*x^2 + 50*x + 25))/(169*x^4 + 390*x^2 + 225) - 1;
can(1) = Ires(a(1))*grad(a(1));
can(2) = Ires(a(2))*grad(a(2));
can(3) = Ires(a(3))*grad(a(3));
end
```

And then solve this system of three identical equations with different initial conditions:

>> [T,A] = ode45(@canonical3, [1,160], [1.15,1.2,1.5]);

>> plot(T,A)

>>



Our model explains evolution from virulent to beneficial (negative virulence)!!!

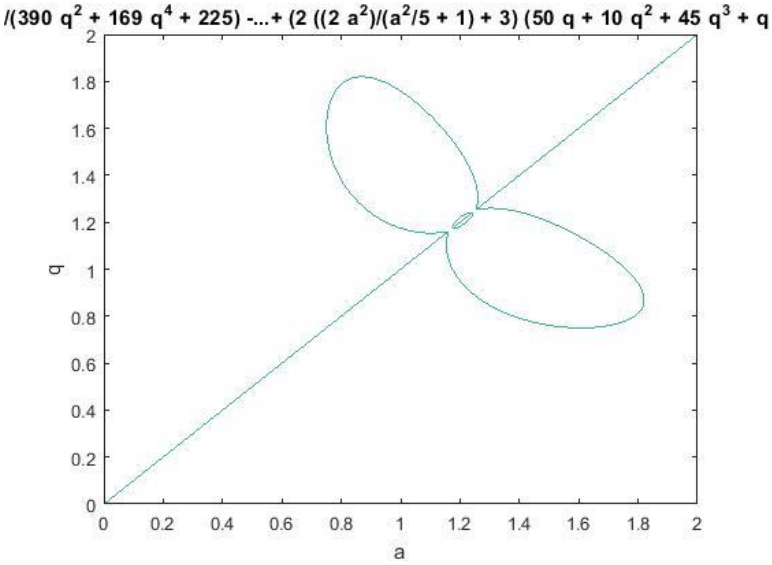
How to draw MIP: it is PIP and its mirror image:

>> ezplot(s==0,[0,2])

>> hold on

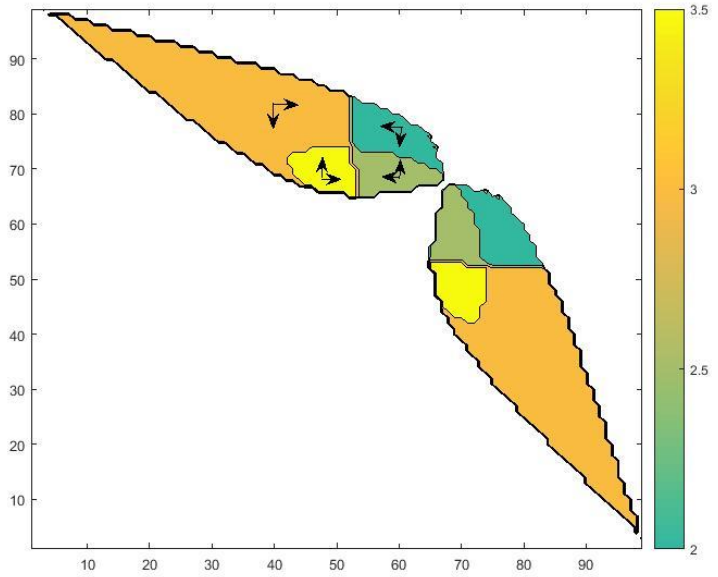
>> ezplot(subs(s, {q,a}, {a,q})==0,[0,2])

>>



Selection gradients can be used to draw a contour plot where one can see isoclines roughly:

```
for i = 1:99
    for j = 1:99
        a = 1;
        b = 1;
        if Grad1(j, i) > 0
            a = 2;
        end
        if Grad2(j, i) > 0
            b = 1.5;
        end
        Data(j, i) = a + b;
    end
end
contourf(Kuvadata, [2, 2.5, 3, 3.5])
```



From that figure could get initial guess for isocline and use numerical continuation with vpasolve.