Cusp Catastrophe How−to

by W. Garrett Mitchener

In this worksheet, I illustrate how to generate a bifurcation diagram for a two−parameter dynamical system in one dependent variable. The particular types of bifurcations that can happen form what's called a *cusp catastrophe*. This worksheet shows several ways to generate the cusp picture that illustrates what happens.

Setup

Let's work with the dynamical system given by:

$$\dot{x} == h + r\,x - x^3$$

First, here's how to define a function for the right hand side:

```
f[x_, r_, h_] = h + r x - x^3
```

$$h + r\,x - x^3$$

First method: Solve for fixed point collision directly

Fixed points happen when $\dot{x} == 0$, and bifurcations happen when fixed points collide (and in other situations). But look what happens when we try to solve for the fixed points:

```
fps = Solve[f[x, r, h] == 0, x]
```

$$\left\{\left\{x \to -\frac{\left(\frac{2}{3}\right)^{1/3} r}{\left(-9\,h + \sqrt{3}\,\sqrt{27\,h^2 - 4\,r^3}\right)^{1/3}} - \frac{\left(-9\,h + \sqrt{3}\,\sqrt{27\,h^2 - 4\,r^3}\right)^{1/3}}{2^{1/3}\,3^{2/3}}\right\},\right.$$

$$\left\{x \to \frac{\left(1 + i\sqrt{3}\right) r}{2^{2/3}\,3^{1/3}\left(-9\,h + \sqrt{3}\,\sqrt{27\,h^2 - 4\,r^3}\right)^{1/3}} + \frac{\left(1 - i\sqrt{3}\right)\left(-9\,h + \sqrt{3}\,\sqrt{27\,h^2 - 4\,r^3}\right)^{1/3}}{2\,2^{1/3}\,3^{2/3}}\right\},$$

$$\left.\left\{x \to \frac{\left(1 - i\sqrt{3}\right) r}{2^{2/3}\,3^{1/3}\left(-9\,h + \sqrt{3}\,\sqrt{27\,h^2 - 4\,r^3}\right)^{1/3}} + \frac{\left(1 + i\sqrt{3}\right)\left(-9\,h + \sqrt{3}\,\sqrt{27\,h^2 - 4\,r^3}\right)^{1/3}}{2\,2^{1/3}\,3^{2/3}}\right\}\right\}$$

Those expressions are pretty nasty, and you can't tell which roots are real or complex. Don't let the i's in those two lower expresssions fool you. The square roots may generate complex numbers depending on r and h that cancel out with the i's on top. For the moment, suppose that's not a problem. Now imagine trying to set two of these equal and solve for r and h:

```
Solve[(x /. fps[[1]]) == (x /. fps[[2]]), h]
```

$$\left\{\left\{h \to -\frac{2\, r^{3/2}}{3\sqrt{3}}\right\}, \left\{h \to \frac{2\, r^{3/2}}{3\sqrt{3}}\right\}\right\}$$

We got lucky: That's the right answer, and the computer didn't crash. I don't recommend this method because if your dynamical system is more complicated than this example, the expressions for the roots will probably be so big and complicated that *Mathematica* can't deal so easily with them. It's worth a try in that case, but there are alternatives. I'll plot the solution below, after showing an alternative method of deriving it.

Second method: Combine fixed point solve with linear stability analysis failure

An alternative is to use the fact that *Mathematica* can handle systems of polynomial equations very easily. Here, we `Solve` for the parameter values such that there is a fixed point at x, and such that linear stability analysis fails there.

```
bifSol = Solve[{f[x, r, h] == 0, D[f[x, r, h], x] == 0}, {r, h}]
```

$$\{\{h \to -2\, x^3,\ r \to 3\, x^2\}\}$$

We want to plot r horizontally and h vertically to reproduce the figure on p. 71 of Strogatz. The nice thing about this result is that it gives h and r very simply in terms of the fixed point x, so we can vary x, and generate a plot of h vs. r parametrically. Here's how to set that up.

Just so you'll know, the solution comes as a list of rule tables, because some systems of equations may have many solutions. So we use bifSol[[1]] to fish out the single rule table we want:

```
bifSol[[1]]
```

$$\{h \to -2\, x^3,\ r \to 3\, x^2\}$$

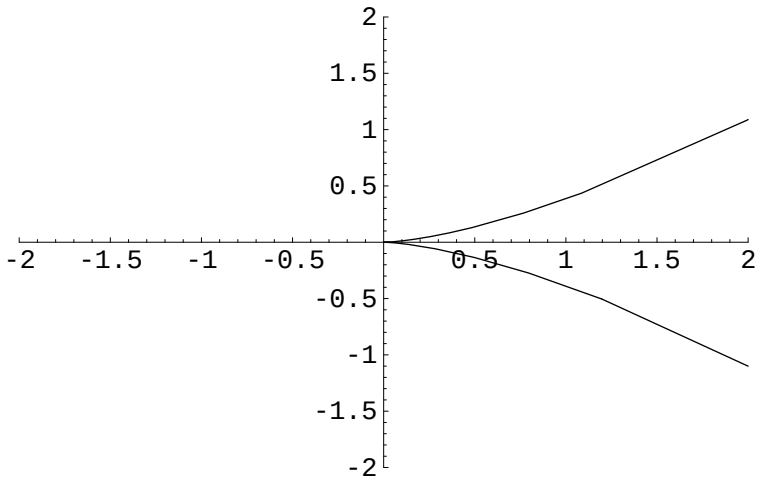
Then we have to make an expression for `ParametricPlot`.

```
{r, h} /. bifSol[[1]]
```

$$\{3\, x^2,\ -2\, x^3\}$$

Finally, here's the plot command:

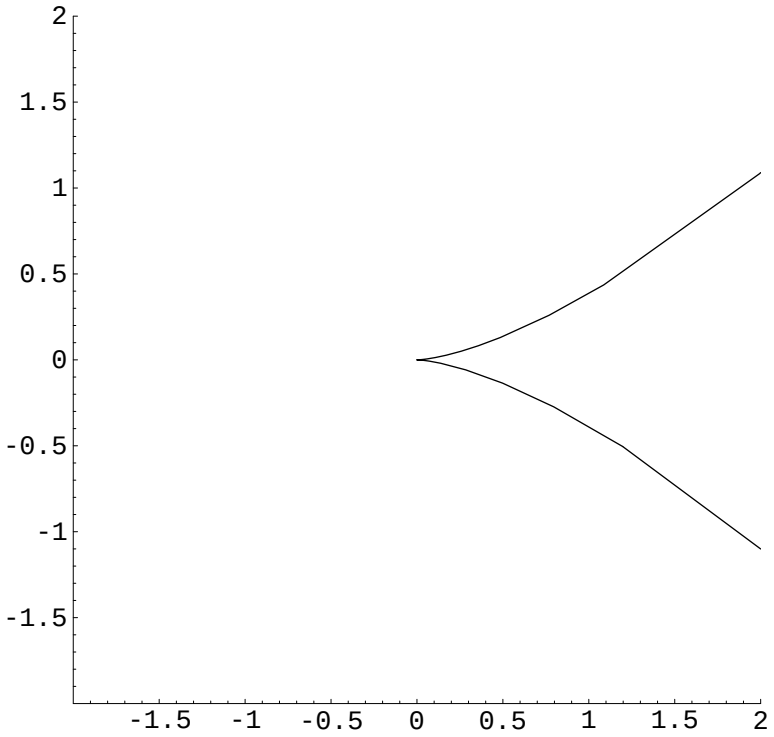
```
ParametricPlot[{r, h} /. bifSol[[1]], {x, -5, 5},
 PlotRange → {{-2, 2}, {-2, 2}}]
```




```
- Graphics -
```

Just so we can see the cusp more clearly, here's a set of plotting options (see 1.9.2) that moves the axes, and sets the aspect ratio so the vertical axis isn't squished:

```
ParametricPlot[{r, h} /. bifSol[[1]], {x, -5, 5},
 PlotRange → {{-2, 2}, {-2, 2}},
 AxesOrigin → {-2, -2},
 AspectRatio → 1]
```




```
- Graphics -
```

By the way, I just guessed that x should go from −5 to 5. Alternatively, you could solve for the fixed points explicitly for $r = 2$ (the right−hand edge) and get an idea for the x range that way.

Third method: Discriminants

Every polynomial has a discriminant. (See `http://mathworld.wolfram.com/PolynomialDiscriminant.html`.) The definition is kind of complicated: You start by numbering all the roots of a polynomial r_1, ..., r_n. Remember, every complex polynomial of degree n has n complex roots. Then the discriminant is defined to be the product of the squares of the differences between every possible pair of roots:

$$D = \prod_{i=1}^{n} \prod_{j=i+1}^{n} (r_i - r_j)^2$$

So the discriminant is zero if two of the roots coincide. Luckily, there's a way to compute the discriminant just from the coefficients of the polynomial. The following incantation defines such a calculation. (Don't worry about how this works for now.)

```
Discriminant[p_?PolynomialQ, x_] := With[{n = Exponent[p, x]},
  Cancel[((-1)^(n (n - 1) / 2) Resultant[p, D[p, x], x]) /
    Coefficient[p, x, n]^(2 n - 1)]]
```

Here's the familiar discriminant for a quadratic polynomial, up to a factor of a^2 :

```
Discriminant[a x^2 + b x + c, x]
```

$$\frac{b^2 - 4\,a\,c}{a^2}$$

So if our function $f[x, r, h]$ undergoes a bifurcation at parameter values r and h, two fixed points collide, which means the discriminant must be zero:

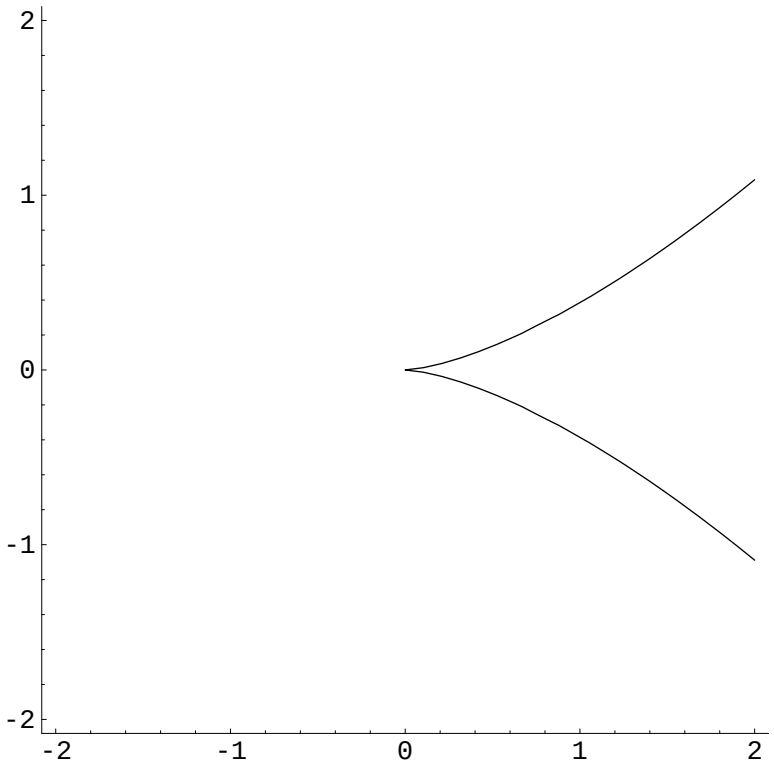
```
discF = Discriminant[f[x, r, h], x]
```

$$-27\,h^2 + 4\,r^3$$

The equation $-27\,h^2 + 4\,r^3 == 0$ defines a curve in the (r, h) implicitly, and we want to get a picture of it. There are different ways to do this. One is to use `ImplicitPlot`. We have to first load a package to make that command available.

```
<< Graphics'

ImplicitPlot[discF == 0, {r, -2, 2}, {h, -2, 2}]
```




```
- ContourGraphics -
```

An alternative is to solve for one variable in terms of the other. This can get a little tricky:

```
hSol = Solve[discF == 0, h]
```

$$\left\{\left\{h \to -\frac{2\,r^{3/2}}{3\sqrt{3}}\right\},\ \left\{h \to \frac{2\,r^{3/2}}{3\sqrt{3}}\right\}\right\}$$

Now we need both solutions. This expression gives us a list of solutions for h:

```
h /. hSol
```

$$\left\{-\frac{2\,r^{3/2}}{3\sqrt{3}},\ \frac{2\,r^{3/2}}{3\sqrt{3}}\right\}$$

The `/.` command can work with a list of rule tables, and produces a list of what happens when you apply the different rule tables to the expression on the left. Just so you see what it's doing:

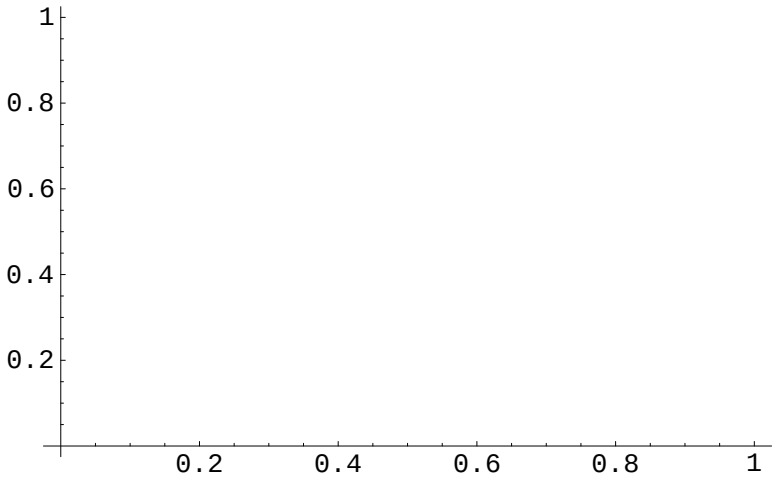
```
Foo[bar, baz] /. {{Foo → f, bar → x, baz → y},
  {Foo → g, bar → a, baz → b}}
```

```
{f[x, y], g[a, b]}
```

Now for our plot. Since we have h as a function of r, we can use the regular `Plot` command, but we need to plot both solutions.

```
Plot[h /. hSol, {r, -2, 2}]
```

```
– Plot::plnr : h /. hSol is not a machine-size real number at r = -2.. More...
– Plot::plnr : h /. hSol is not a machine-size real number at r = -1.83773. More...
– Plot::plnr : h /. hSol is not a machine-size real number at r = -1.66076. More...
– General::stop :
   Further output of Plot::plnr will be suppressed during this calculation. More...
```



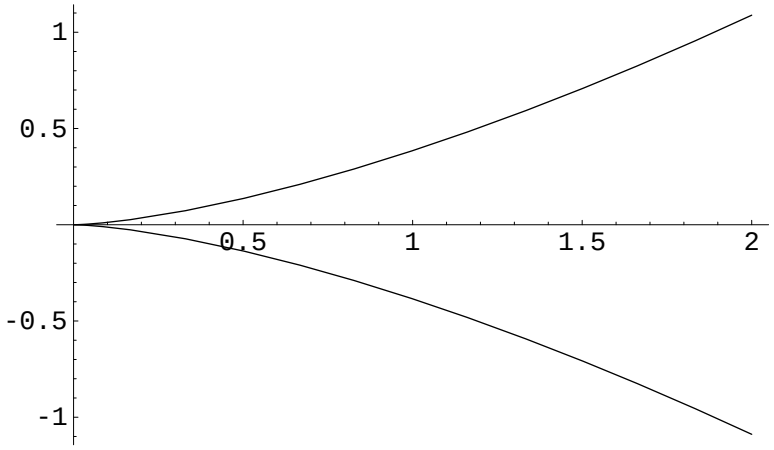

```
- Graphics -
```

That isn't good. What sometimes happens is that *Mathematica* evaluates the expression you're plotting in a funny order, so for example, it sets $r = -1.66$, and looks at h /. hSol but doesn't evaluate the `/.` substitution, so

there's nowhere to plug in r, and it gets confused. What usually works in this case is to tell *Mathematica* not to delay evaluating the expression to be plotted. You do that by wrapping it in `Evaluate`. This is a good trick to know. `Solve`, `Plot`, and other functions are documented as having this odd behavior: They "evaluate their arguments in a non−standard way" which is a clue that you might need to use `Evaluate`. Anyway, here's the repaired `Plot` command:

```
Plot[Evaluate[h /. hSol], {r, -2, 2}]
```

```
– Plot::plnr : -(2 r^<<1>>)/(3 Sqrt[3]) is not a machine-size real number at r = -2.. More...
– Plot::plnr : -(2 r^<<1>>)/(3 Sqrt[3]) is not a machine-size real number at r = -1.83773. More...
– Plot::plnr : -(2 r^<<1>>)/(3 Sqrt[3]) is not a machine-size real number at r = -1.66076. More...
– General::stop :
   Further output of Plot::plnr will be suppressed during this calculation. More...
```




```
- Graphics -
```

By the way, you can ignore the complaints about not getting a real number. The plotting process complains because the expression generates complex numbers for $r < 0$ and nothing bad happens; they just don't show up in the plot.

We can also solve for r as a function of h, which happens to be easier in this case.

```
rSol = Solve[discF == 0, r]
```

```
– General::spell1 : Possible spelling error: new
    symbol name "rSol" is similar to existing symbol "hSol". More...
```

$$\left\{\left\{r \to 3\left(-\frac{1}{2}\right)^{2/3} h^{2/3}\right\}, \left\{r \to \frac{3\,h^{2/3}}{2^{2/3}}\right\}, \left\{r \to -\frac{3\,(-1)^{1/3}\,h^{2/3}}{2^{2/3}}\right\}\right\}$$

Two of the solutions are complex for every h, so the only one we need to worry about is:

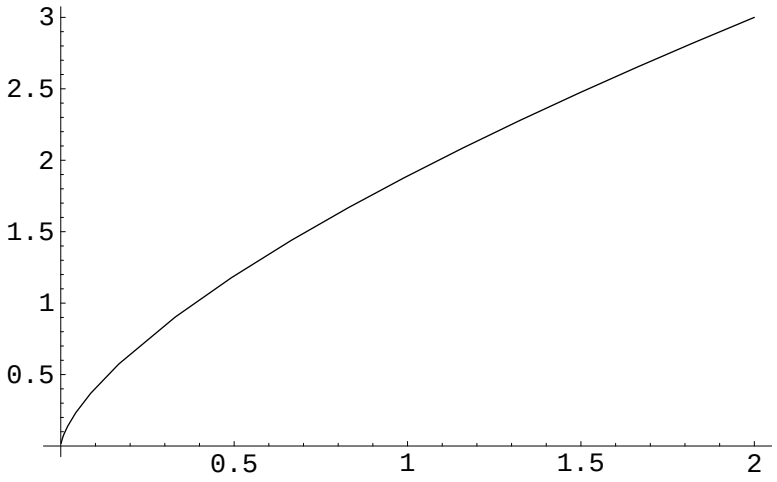
```
rSolReal = 3 h^(2/3) / 2^(2/3)
```

$$\frac{3\,h^{2/3}}{2^{2/3}}$$

And here I've just used the mouse to select the one I want from the output, and copied it to an input cell. Now it should be clear why we get a cusp: Functions of the form $(x^2)^{1/n}$ for $n > 2$ have cusp−shaped graphs. The plot is a little weird:

```
Plot[rSolReal, {h, -2, 2}]
```

```
– Plot::plnr : rSolReal is not a machine-size real number at h = -2.. More...
– Plot::plnr : rSolReal is not a machine-size real number at h = -1.83773. More...
– Plot::plnr : rSolReal is not a machine-size real number at h = -1.66076. More...
– General::stop :
   Further output of Plot::plnr will be suppressed during this calculation. More...
```




```
- Graphics -
```

The left hand side is missing and we get complaints about not getting real numbers for negative values of h. That's because *Mathematica* is extremely careful with complex numbers, and knows that $x^{p/q}$ is generally not well defined for negative x. The case of $\frac{p}{q} == \frac{1}{2}$ is an obvious example. So we end up getting only half the plot we wanted. The parametric form is better because it doesn't have this problem.

Fourth method: Template polynomials

Just so you'll know, "template polynomial" is a term I made up. As far as I know, there's no standard name for this technique.

We suppose that a bifurcation occurs at (r, h) such that two fixed points coincide. That means there's a double root. So, to trace out such values of r and h, we set our f equal to a polynomial with the same degree, and same highest−power coefficient, but with overlapping roots. That's what I call the template:

```
template = -(x - p1)^2 (x - p2)
```

```
-(-p1 + x)^2 (-p2 + x)
```

Here, p1 and p2 are the two roots, and by putting in $(x - \text{p1})^2$, I've specified that p1 is a double root. I put in the $-$ sign because the highest−order term of f is $-x^3$ and the template has to match that. To make use of this template, what we need to do is equate the coefficients of our template to the coefficients of f. That gives a bunch of equations for r and h. We can get the coefficients by using the `CoefficientList` function:

```
CoefficientList[a x^2 + b x + c, x]
```

```
{c, b, a}
```

```
CoefficientList[f[x, r, h], x]
```

```
{h, r, 0, -1}
```

```
CoefficientList[template, x]
```

```
{p1^2 p2, -p1^2 - 2 p1 p2, 2 p1 + p2, -1}
```

We now need to turn these into a list of equations. The `Thread` function is used to turn a function of two lists into a list of functions of pairs:

```
Thread[f[{a, b, c}, {d, e, f}]]
```

```
{f[a, d], f[b, e], f[c, f]}
```

And it works with equations, too:

```
Thread[{a, b, c} == {d, e, f}]
```

```
{a == d, b == e, c == f}
```

So we can equate our coefficients with the incantation:

```
coefficientEqs =
 Thread[CoefficientList[f[x, r, h], x] == CoefficientList[template, x]]
```

```
{h == p1^2 p2, r == -p1^2 - 2 p1 p2, 0 == 2 p1 + p2, True}
```

Notice that we get a couple of easy equations. First, the highest order coefficients are both -1 because I defined them that way, so the last entry is $-1 == -1$ which simplifies to `True`. Second, we get $\text{p2} = -2\,\text{p1}$ out of the next highest, which is very nice. That let's us get rid of p2 entirely. This is easy enough to do by hand in this case, but *Mathematica* can also do it automatically. We use the `Eliminate` function to get rid of p2.

```
coefficientEqs2 = Eliminate[coefficientEqs, p2]
```

```
h == -2 p1^3 && r == 3 p1^2
```

I'd really rather have a list:

```
coefficientEqs3 = coefficientEqs2 /. p_ && q_ → {p, q}
```

```
{h == -2 p1^3, r == 3 p1^2}
```

This is the same set of parametric equations we got before, so I won't plot them again.

Incidentally, suppose we wanted to find the parameter values where all three fixed points collide. That's the pitchfork bifurcation. Then we simply use a different template:

```
template2 = -(x - p)^3
```

```
-(-p + x)^3
```

Now p is a triple root. We proceed as before, equating coefficients:

```
coefficientEqsTriple =
 Thread[CoefficientList[f[x, r, h], x] == CoefficientList[template2, x]]
```

```
{h == p^3, r == -3 p^2, 0 == 3 p, True}
```

Now things are much more restricted:

```
Eliminate[coefficientEqsTriple, p]
```

```
h == 0 && r == 0
```

By the way, I hope this last solution is obvious from the system of equations.