Background for
"Bifurcation, Catastrophe, Singularity, and All That"

https://people.sc.fsu.edu/~jburkardt/presentations/bifurcation_background.pdf

.....

John Burkardt

21 October 2008

http://people.scs.fsu.edu/~burkardt/latex/bifurcation/bifurcation_background.pdf

1.) *The Buckling Spring*: Two straight springs and a circular spring are arranged as in Figure 1. The two straight springs are naturally of length 1, but are currently length **L**. The left spring has an endpoint fixed at the origin, and the angle that the left spring makes with the x-axis is θ. The right spring endpoint is free to move horizontally.

A horizontal load λ is applied at the right spring endpoint; a vertical load μ is applied at the point where the springs meet. Finally, there is a spring force *between* the two straight springs, applied by the circular spring, which tends to draw them together.

If the system is in equilibrium, the balance of forces is:

$$-2(1-L) + 2\lambda \cos\theta + \mu \sin\theta = 0$$
$$0.5\theta - 2\lambda L \sin\theta + \mu L \cos\theta = 0$$

These equations can be solved for λ and μ in terms of L and θ:

$$\lambda(L,\theta) = (1-L)\cos\theta + 0.25\theta \sin\theta / L$$
$$\mu(L,\theta) = 2(1-L)\sin\theta - 0.5\theta \cos\theta / L$$

To study the structure of solutions, we can vary the fundamental parameters **L** and θ and consider the pattern of the associated variables λ and μ.

2.) *The Freudenstein Roth Function*: Suppose we are looking for a solution (x_1, x_2) of the following pair of nonlinear equations:

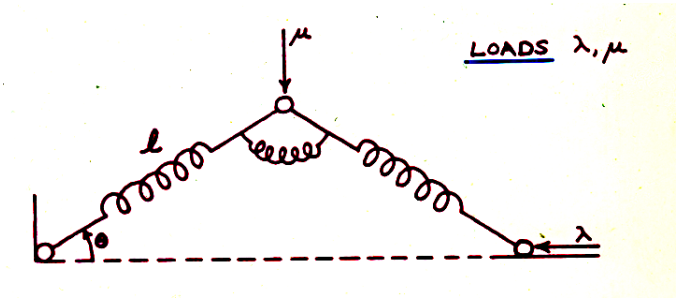


Figure 1: The Buckling Spring

$$x_1 - x_2^3 + 5x_2^2 - 2x_2 - 13 = 0$$
$$x_1 + x_2^3 + x_2^2 - 14x_2 - 29 = 0$$

symbolized by $g(x_1, x_2) = 0$. A standard way to solve such a system is called *Newton's method*. It allows you to pick an arbitrary starting guess, and then it carries out an iteration to seek an approximate solution. The problem is, Newton's method often won't work unless the starting guess is close to the correct solution. But if we don't know the solution in the first place, how are we supposed to come up with an approximate solution that's good enough? And if we guess an approximate solution but Newton says it isn't good enough, what can we do?

One devious way to get Newton's method to solve your nonlinear equations is to pick *any* pair of values (x_1^*, x_2^*) as a starting guess, add a third variable x_3, and write a new set of equations $f(x_1, x_2, x_3)$

$$f(x_1, x_2, x_3) = g(x_1, x_2) + (x_3 - 1)g(x_1^*, x_2^*)$$

with the properties that

- When $x_3 = 0$, $f(x_1^*, x_2^*, 0) = 0$, so we have a starting point;
- When $x_3 = 1$, then if we have a solution to $f(x_1, x_2, 1) = 0$, it is also true that $g(x_1, x_2) = 0$.

The only reason to add this complication is that we can take steps, *as small as we like* to nudge x_3 from the known solution at 0 to 1, where our unknown and desired solution is. And as long as each nudge of x_3 doesn't affect the solution too much (oops, could there be a problem here?) we can use our current solution as a starting guess for the solution at the new value of x_3. This idea is known as *the continuation method.*

Of course we can pick any starting point; we hope that for any starting point, we end up at the same final answer (but this is not guaranteed!). It is possible that some starting points result in a much longer "journey" to the final answer! And it may also be possible that, along the way, we can't always use x_3 as the variable that we increase each time, if the solution curve decides to bend back on us!

3.) *The Aircraft Model*: a simple model of an aircraft keeps track of 5 state variables, *roll, pitch, yaw, angle of attack*, and *sideslip*, describing the attitude of the plane, and 3 control variables, *elevator, aileron*, and *rudder* which are controls the pilot can use to try to adjust the attitude of the plane.

This particular plane, by the way, is flying at Mach 0.9, at a height of 20,000 feet.

While the effects of the controls are usually predictable, there are cases where a a sequence of slight increases to a control cause a corresponding small change in the attitude, until one more slight increase causes the plane to "jump", with large changes to the state variables.

The equilibrium equations can be written as $A * x + \phi(x) = 0$, where A is a 5x8 matrix:

$$\mathbf{A} = \begin{pmatrix} -3.933 & 0.107 & 0.126 & 0 & -9.99 & 0 & -45.83 & -7.64 \\ 0 & -0.987 & 0 & -22.95 & 0 & -28.37 & 0 & 0 \\ 0.002 & 0 & -0.235 & 0 & 5.67 & 0 & -0.921 & -6.51 \\ 0 & 1 & 0 & -1 & 0 & -0.168 & 0 & 0 \\ 0 & 0 & -1 & 0 & -0.196 & 0 & -0.0071 & 0 \end{pmatrix}$$

and $\phi(x)$ is a set of quadratic terms:

$$\phi(\mathbf{x}) = \begin{pmatrix} -0.727x_2x_3 + 8.39x_3x_4 - 684.4x_4x_5 + 63.5x_4x_7 \\ 0.949x_1x_3 + 0.173x_1x_5 \\ -0.716x_1x_2 - 1.578x_1x_4 + 1.132x_4x_7 \\ -x_1x_5 \\ x_1x_4 \end{pmatrix}$$

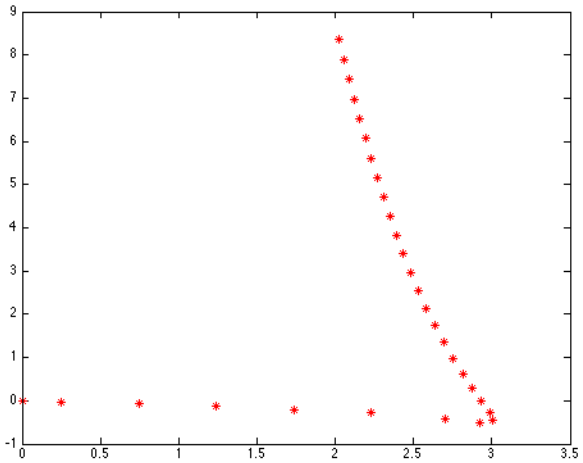


Figure 2: Evidence of a limit point

We have three remaining degrees of freedom. It is natural to use two of them to specify fixed values of control parameters, and leave one control parameter free to vary. For the systems we are interested in, our conditions will be

$$x_6 = \textbf{elevator value}$$
$$x_7 = variable$$
$$x_8 = 0$$

So in this model, we will assume that the pilot is free to adjust the aileron x_7, but the other two controls are fixed. The set of solutions form a curve. For neighboring points on the curve, the values of x_7 will be close. So if we imagine traveling along the curve, we are essentially tapping on the aileron control, increasing it a little as we move from point to point.

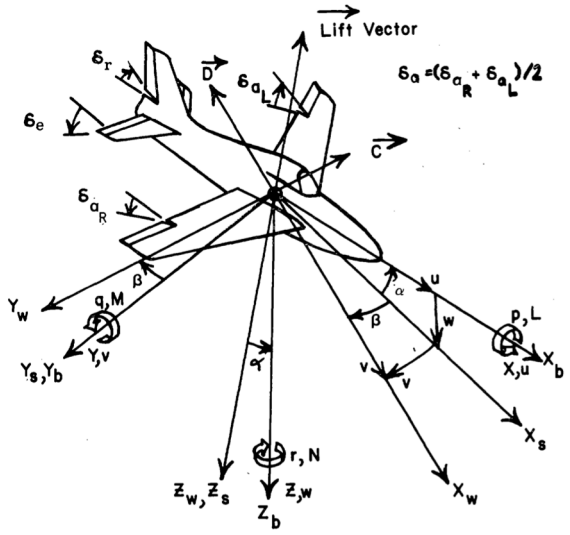
However, when the pilot actually flies the plane, sometimes a small increase in the aileron value causes the plane's behavior to jump discontinuously. How can this be? It must mean that, for the particular control settings, we've reached a *limit point* in x_7, that is, a place where the solution curve bends back. When the pilot increases the aileron setting, the plane essentially has to skip the whole part of the curve that has bent back, and jump to a new section that is relatively far away.

So the existence and location of limit points on a solution curve can tell us something important.

To hunt for limit points, we specify a fixed **elevator value**. Our basic computation now follows the solution curve, generating a sequence of points. However, this time, we are constantly watching for limit points in the variable x_7, that is, points on the curve in which x_7 changes from increasing to decreasing (or vice versa).

We can detect that we've passed a limit point simply by observing successive values of x_7, or by comparing the corresponding values of the tangent vector, tan_7. The tangent vector is telling us how each variable is changing along the curve. When tan_7 is 0, that generally means that x_7 has reached a limit point.

To actually get a good approximation to the exact location of the limit point, we need to do a special iteration that seeks the value of x_7 for which tan_7 goes to zero.



Assignment:

1.) *The Buckling Spring*: Define $\mathbf{M} = 101$ equally spaced values

$$0.25 <= L_i <= 1.75$$

and $\mathbf{N} = 101$ equally spaced values

$$\frac{-3\pi}{8} <= \theta_j <= \frac{+3\pi}{8}$$

Then define $\mathbf{M}$ by $\mathbf{N}$ arrays $\lambda_{i,j}$ and $\mu_{i,j}$ by evaluating the formulas that relate λ and μ to $\mathbf{L}$ and θ.

Use the MATLAB **plot** command on the pair of arrays, to create an image of the structure of the solutions of the buckling spring. Use the MATLAB **axis** command to restrict the plot to $0.1 \leq \lambda \leq 0.9$ and $-0.07 \leq \mu \leq +0.07$.

Turn in 1 plot.

2.) *The Freudenstein Roth Function*: Use the code **p01_target_test.m**, First use the starting point (4,3,0), and compute a series of solutions until you reach the target value (?.?,1). Modify the code to save each computed point in an array so that at the end, you can use the MATLAB command **scatter3** to plot the 3D set of points.

Repeat the exercise, but use the starting point (15,-2,0). Again, use the MATLAB command **scatter3** to plot the 3D points.

Turn in 2 plots.

3.) *The Aircraft Model*: Use the code **p06_limit_test.m**, and do a limit point search, using each of the following values of X6 (**elevator setting**): -0.050, -0.008, 0.000, +0.050, and 0.100. A paper by Melhem and Rheinboldt observed 1, 3, 2, 1, and 1 limit points respectively. Save the points calculated. For each run, make a plot of the values of X1 (**roll rate**) versus X7 (**aileron**). You might notice one extra limit point when you repeat their calculations.

Turn in 5 plots.