Cycle Searching through Relaxation

Jimmy Corno

School of Physics

Georgia Institute of Technology

Atlanta, GA 30332-0430, U.S.A

April 29, 2004

Abstract

A painfully simple method for finding cycles in time independent flows is generalized and demonstrated with the Rossler system.

1 Introduction

Finding periodic orbits in a constant vector field by means of the Newton-Raphson method is very reliable, but in high dimensional flows it becomes impractical. A possible alternative is a relaxation method that relies on minimization of a cost function. This method would essentially be an extension of a multishooting method. Instead guessing of a group of points on a cycle, one starts with an entire guess cycle.

For a periodic orbit, the loop velocity $\tilde{v}$ would be the same as the local flow velocity at every point on the loop. It is therefore natural to define the cost function by the difference between the two.

$$F^2 \equiv \oint_L dt(\vec{v} - \tilde{v})^2 \tag{1}$$

This is fairly straightforward, but the time integral is problematic, since it depends on the parametrization of the loop. This problem can be solved by defining the component of the loop velocity that is parallel to the local flow velocity to also have the same magnitude as the local flow velocity. In other words,

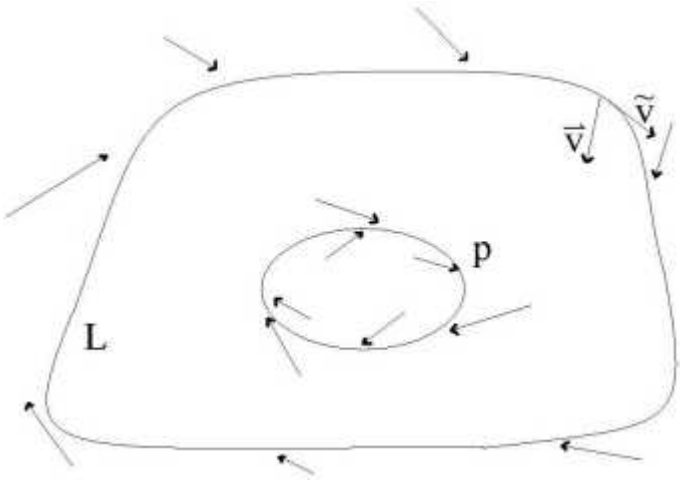


Figure 1: A guess loop L near an actual cycle p.

$$\tilde{v} = \tilde{v}_{\parallel} + \tilde{v}_{\perp} = \vec{v} + \tilde{v}_{\perp} \tag{2}$$

.

This has the added benefit of simplifying the integrand. Unfortunately, it will introduce another problem, which I will address later.

With this new parametrization, the time integral becomes a line integral.

$$F^2 \equiv \oint_L \frac{d\tilde{x}}{|\tilde{v}|}(\tilde{v}_{\perp})^2 \tag{3}$$

which simplifies even further.

$$F^2 = \oint_L dx|v(\sec\theta - \cos\theta)| \tag{4}$$

Here θ is the angle between the loop velocity and the flow velocity. The cost function should now be easy to calculate. If the loop is broken into straight segments $\vec{l}$ then we can use

$$\cos\theta = \frac{\vec{v}\cdot\vec{l}}{vl} \tag{5}$$

Reduction of the cost function is extremely simple and requires very little additional calculation. Each point on the loop is adjusted based on the integral of $\tilde{v}_\perp$.

$$\Delta\tilde{x}_i = \gamma \int_{x_{i-1}}^{x_i} dt\tilde{v}_\perp \tag{6}$$

where $\tilde{x}_i$ is a guess point on L and γ is a damping constant to reduce overshooting. In my program γ is adjusted based on local contribution to the cost function so that especially bad regions of the loop converge more quickly.

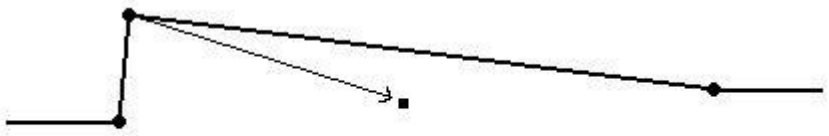
2 Implementation

Although the method appears to work well, its limitations quickly became apparent even in two dimensions. Most importantly, it fails if, at any point in the loop, the loop velocity is perpendicular or antiparallel to the flow velocity (the problem I mentioned earlier), which means time either stops or moves backwards. I can't see any way to overcome this limitation without completely reworking the method, so the initial guess must at least be vaguely in the direction of the flow at all points.

My relaxation was too simple even for the two dimensional case because it wasted a lot of information. Instead of moving the endpoint of each loop segment, it is much better to move both ends of the segment at the same time. Deformations in the loop propagate through much more quickly this way, resulting in nearly an order of magnitude fewer calculations.

In three dimensions the biggest problem was kinking and bunching of the loop. If allowed to move freely, the points that define the loop will move together in some areas, and one will inevitably move so that the loop turns back on itself. At this point, if the program is simply looking for a decrease in the cost function, it runs out of control. This problem originally led me to believe that the method was unworkable, as every trajectory mysteriously went wrong no matter how close to an actual cycle. But the fix was remarkably simple.

Whenever the length ratio of neighboring sections grew beyond a certain point, the middle point was replaced with the average of the two outer points. The same method was used to remove kinks, which were located by their negative contributions to the cost function.



Unkinking the loop by averaging the offending point.

The program won't work without this correction, but there is probably a much better way to implement it than this. The curvature of the line is removed at this point, which could probably be avoided by taking into account points beyond the immediate neighbors. It's also difficult to find a compromise between speed and effectiveness for the bunching threshold; I had to rely on trial and error. Adjustments based on local curvature of the vector field would be a big help.

3 Results

I was able to find three dimensional results, but only in the simplest case I could find (most of my time was spent working on that bunching problem). I found a stable periodic orbit for the Rossler system, roughed it up, and then let the program track it back down.

$$\begin{aligned} \dot{x} &= -y - z \\ \dot{y} &= x + ay \\ \dot{z} &= b + z(x + c) \end{aligned} \tag{7}$$

I used $a = b = 0.2$ and $c = 2.5$ to find a period 1 cycle (only one loop). The program was much more effective than I expected it to be.

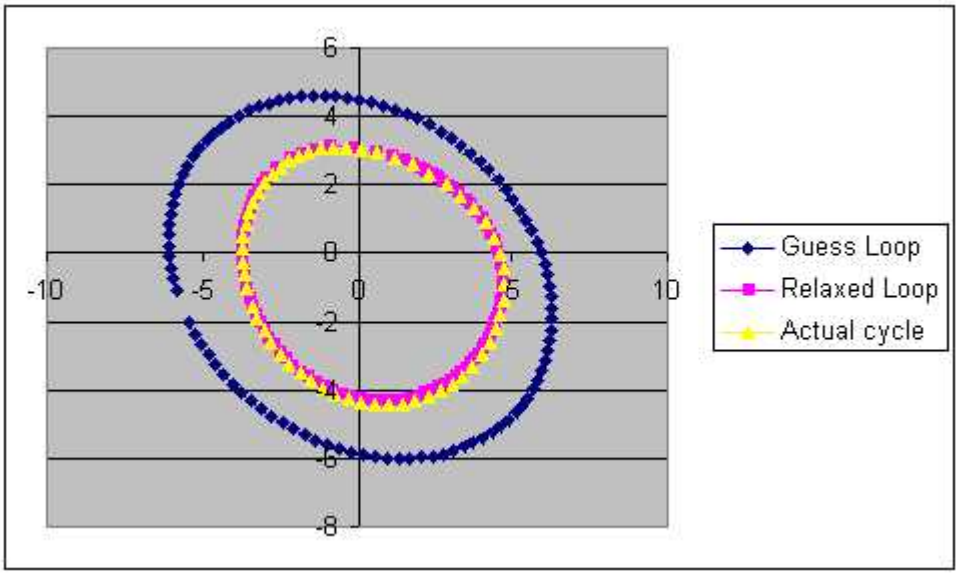


Figure 2: X-Y projection of a Rossler system cycle.

This took approximately 10,000 iterations to achieve (a little less than one minute) with 100 loop points. The X-Z and Y-Z projections are similarly accurate.

4 Conclusions

I am fairly confident that the program would work very well on more complicated systems such as variations of the Rossler system. Given another day or so I could probably fit higher period Rossler flows (there is a period two cycle for $c = 3.5$ and a period three for $c = 6.7$) without any major changes to the program. However, for anything more complicated would be helped by some modifications. I've already mentioned the improvement for the unbunching and unkinking. I believe it would also be useful to reevaluate the line integration. I simply considered the loop to be constructed of straight lines joining the guess points, adn I integrated along those lines. A more respectable method would

take the curvature of the loop into account and get a more accurate assessment of the cost function. And though I cannot think of any better method, I believe there must be a better way to approach the relaxation. The integration of $\tilde{v}_\perp$ seems much too simplistic, and it may be beneficial to take into account more neighboring points in calculating the loop deformation.