Pseudo Arclength Continuation

Niall Hanevy

May 15, 2023

Introduction

These notes are mainly based on the derivation in Scott's thesis with comments about how each step was implemented in the Matlab file in the adjoining folder. The code attached is for the Blasius problem but I am optimistic that it should work for problems more generally.

Pseudo Arclength Continuation

Given a polyomial eigenvalue problem defined as

$$\mathcal{D}(\alpha;\omega,\mathrm{R})\phi := \left(\alpha^2\boldsymbol{A}_2 + \alpha\boldsymbol{A}_1 + \boldsymbol{A}_0\right)\phi = 0, \tag{1}$$

where (α,ϕ) form and eigenvalue eigenvector pair for given values of ω and R. We seek solutions of equation (1) such that $\Im(\alpha) = 0$. A standard continuation scheme would work as follows. Take inital known solution of equation (1) so that $\alpha_i(\omega_0,\mathrm{R}_0) = 0$. From there define a step length $\mathrm{d}\,\mathrm{R}$ and seek a new solution $\alpha_i(\omega,\mathrm{R}_0+\mathrm{d}\,\mathrm{R}) = 0$. Where ω is updated using Newtons method until the solution has sufficiently converged. Thd issue with this approach is that the regardless of how small $\mathrm{d}\,\mathrm{R}$, the scheme will eventually fail to converge when a turning point is met.

It is for this reason that we introduce an additional arclength parameter s and parameterise ω and R with s so that we now seek solutions of the form

$$\alpha_i(\omega(s),\mathrm{R}(s)) = 0, \tag{2}$$

assuming that ω and R depend smoothly on s we can differentiate equation (2) with resect to s to get

$$\frac{\partial\alpha_i}{\partial\omega}\dot{\omega} + \frac{\partial\alpha_i}{\partial\,\mathrm{R}}\dot{\mathrm{R}} = 0, \tag{3}$$

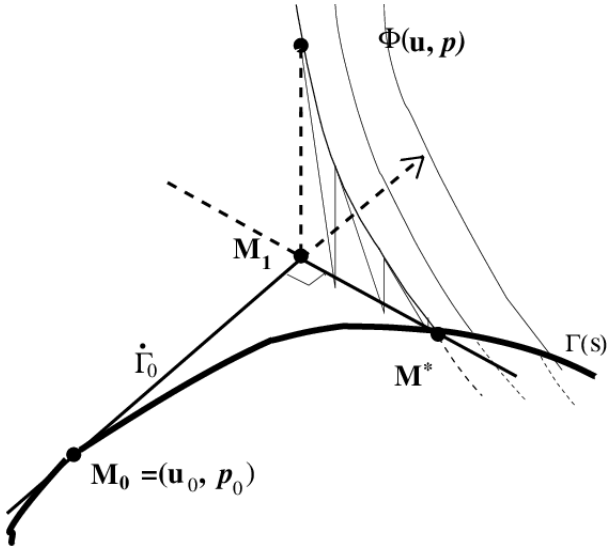

Figure 1: Schematic of psuedo arclength continuation taken from researchgate

with the dot notation being used to denote differentiation with respect to s. These derivatives are contrained by the normalisation requirement

$$|\dot{\omega}|^2 + |\dot{\mathrm{R}}|^2 = 1. \tag{4}$$

This condition along with equaiton (2) will be used to form the system of equations which we will use to solve all ω and R o the neutral curve. The normalisation condition is approximated by

$$N(\omega, \mathrm{R}, s) = \begin{bmatrix} \dot{\omega}_0 \\ \dot{\mathrm{R}}_0 \end{bmatrix} (\omega\omega_0, \mathrm{R} - \mathrm{R}_0) - \mathrm{d}s = 0. \tag{5}$$

This condition states that the new point must lie on a plane orthongonal to the tangent vector at the last known point, at a distance ds from (ω_0, R_0). This is shown in figure 1 (with different notaion). The full system is thus

$$\begin{bmatrix} \alpha_i(\omega, \mathrm{R}) \\ N(\omega, \mathrm{R}, s) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

This is solved for the new point (ω, R) specifying ds. In the code $\mathrm{d}s = \frac{\omega_c \dot{\mathrm{R}}_0}{r_c \dot{\omega}_0}$ where $r_c = 200$ and $\omega_c = 0.1$ are scales for R and ω repecively. This resulted in reasonable step sizes for the Blasius problem though may need to be adjusted for other problems.

Issues

In the code if Newtons method does not convege after 4 iterations ds is halved. This is due to the "hump" in the upper branch which was not satisfactorily resolved. Also where the "hump" flatttens out the curve is locally very flat and so a large step size is taken far beyond the critical value, hence the values chosen. I suspect that for other problems this may need some tweaking to get to work properly but the code works for both the upper and lower branches as is.

Calculating Derivatives

The method for calculating derivatives is mentioned both in Scotts thesis and Alexander Ramage's, however both cite the Bridges and Morris paper. All of these are in the papers folder. Here I will outline the approach.

Starting with the orignal dispersion relation

$$\mathrm{D}(\alpha;\omega\,\mathrm{R})\phi_R = 0$$

The matrix D is calcuated by calling

```
[A0, A1, A2] = D(U,dU,Y_max, delta, N, S, dS, d2S, R, omega)
 DD = A0 + e1*A1 + e1^2*A2;
```

(orignally *mat(...)*, apologies for changing the names uneccessarily). where *e1* is the eigenvalue for a given (ω_0, R_0).

To calculate the eigenvectors and inverse power method is used where

$$\phi_R^{i+1} = \frac{D^{-1}\phi_R^i}{||D^{-1}\phi_R^i||}$$

with ϕ_R^0 set equal to a vector of ones. The left eigenvector is calulated in the same fashion by complex conjugate transposing the matrix D so that $D^+\phi_L = 0$

Differentiating the disperson relation with respect to ω and R yields

$$\begin{aligned}\frac{\partial \mathrm{D}}{\partial \omega}\phi_R + \mathrm{D}\frac{\partial \phi_R}{\partial \omega} &= 0,\\ \frac{\partial \mathrm{D}}{\partial \mathrm{R}}\phi_R + \mathrm{D}\frac{\partial \phi_R}{\partial \mathrm{R}} &= 0.\end{aligned}$$

Both of these expressions can be multiplied by the left eigenvector ϕ_L to give

$$\begin{aligned}\phi_L\frac{\partial \mathrm{D}}{\partial \omega}\phi_R &= 0,\\ \phi_L\frac{\partial \mathrm{D}}{\partial \mathrm{R}}\phi_R &= 0,\end{aligned}$$

which (somehow) gives (Bridges and Morris)

$$\begin{aligned}\frac{\partial \alpha_i}{\partial \omega} &= -\frac{\phi_L \frac{\partial \mathrm{D}}{\partial \omega}\phi_R}{\phi_L \frac{\partial \mathrm{D}}{\partial \alpha}\phi_R},\\ \frac{\partial \alpha_i}{\partial \mathrm{R}} &= -\frac{phi_L \frac{\partial \mathrm{D}}{\partial \mathrm{R}}\phi_R}{\phi_L \frac{\partial \mathrm{D}}{\partial \alpha}\phi_R}\end{aligned}$$

using these expressing calculated in the code in *D_lambda* where $\frac{\partial \alpha_i}{\partial \omega} =$ *lambda_w* and $\frac{\partial \alpha_i}{\partial \mathrm{R}} =$ *lambda_R*,

we can calculate $\dot{\omega}_0$ and $\dot{\mathrm{R}}_0$ using

$$\begin{aligned}\frac{\mathrm{d}\alpha_i}{\mathrm{d}s} = 0 \implies \dot{\omega} &= \frac{\frac{\partial \alpha_i}{\partial \mathrm{R}}}{\frac{\partial \alpha_i}{\partial \omega}}\dot{\mathrm{R}},\\ \dot{R} &= \pm\sqrt{1-|\dot{\omega}|^2}.\end{aligned}$$

Or in code

```
lambda_r =  -imag((VL'*AR*VR)/(VL'*Aa*VR));
lambda_w = -imag((VL'*Aw*VR)/(VL'*Aa*VR));

R_s = dir*sqrt(1 -  abs(lambda_r/lambda_w)^2);
w_s = -dir*lambda_r /lambda_w;
```

where *dir* $= \pm 1$ is a paramter which controls the direction travelled along the curve.

Note that in the Newton iteration described in the next section the eigenvectors and tangents *(R_s, w_s)* are only calculated at the previous known point before entering the loop. This is useful for the eigenvectors as they invlove matrix inversions that only have to be performed once at each step rather than being updated at each Newton iteration. However the matrices $\frac{\partial \mathrm{D}}{\partial \mathrm{R}}$ for example are updated at each iteration in Newtons method.

Newtons Method

Recall the definition of H from above

$$H = \begin{bmatrix}\alpha_i(\omega, \mathrm{R})\\ N(\omega, \mathrm{R})\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}.$$

To iterate we first calculate the Jacobian (again using *D_lambda*, using eigenvectors calcuated from previous point on the curve)

$$J = \begin{bmatrix} \frac{\partial \alpha_i}{\partial \omega} & \frac{\partial \alpha_i}{\partial \mathrm{R}} \\ \dot{\omega}_0 & \dot{\mathrm{R}}_0 \end{bmatrix}.$$

The guess is iteratively updated via

$$\begin{bmatrix} \omega^{i+1} \\ \mathrm{R}^{i+1} \end{bmatrix} = \begin{bmatrix} \omega^{i} \\ \mathrm{R}^{i} \end{bmatrix} - J^{-1} H$$

This is iterated until $\max |H| < 10^{-8}$, although this can be altered depending in the desired accurracy. In most cases the solution converges within 3 iterations.

Initial Guess

The tangents from a previous pair of known points are used to calculate the first guess for the new values of (ω, R) This is done by setting

$$\begin{bmatrix} \omega \\ \mathrm{R} \end{bmatrix} = \begin{bmatrix} \omega_0 \\ \mathrm{R}_0 \end{bmatrix} + \mathrm{d}s \begin{bmatrix} \dot{\omega}_0 \\ \dot{\mathrm{R}}_0 \end{bmatrix}$$

Running the Code

In order to change the code for a new problem you would need.

- Base flow solutions
- copy and paste old mat.m into D.m
- update the *D_alpha* , *D_omega* and *D_R* files by taking termwise derivatives of each element in you old mat.m file.
- Alter scaling of ds.

Having compared running times for Blasius with $N = 50$ I saw a 5 times improvement disabiling all plotting and printing. From ~ 400 seconds to ~ 80 seconds. I also managed to get it to plot the whole curve in one go by imposing that

```
if  abs(x1(1) - x0(1)) > 0.1
    dir = - dir;
    x1  = x0;
    x1(1) = x1(1) + 1e-3*sign(w_s);
    R = x1(2);
    omega = x1(1);
    [A0, A1, A2] = D(U,dU,Y_max, delta, N, S, dS, d2S, x1(2), x1(1));
    alpha = filter_eigs(polyeig(A0, A1, A2 ), x1(1) );
    return
end
```

inside the *Newt_iter* function with *x1(1)* being the new value of ω. This seems to work fine in this case but I would like to check the robustness of this rule in other cases (*i.e.* rotating disk type flow) and may need to be modified.

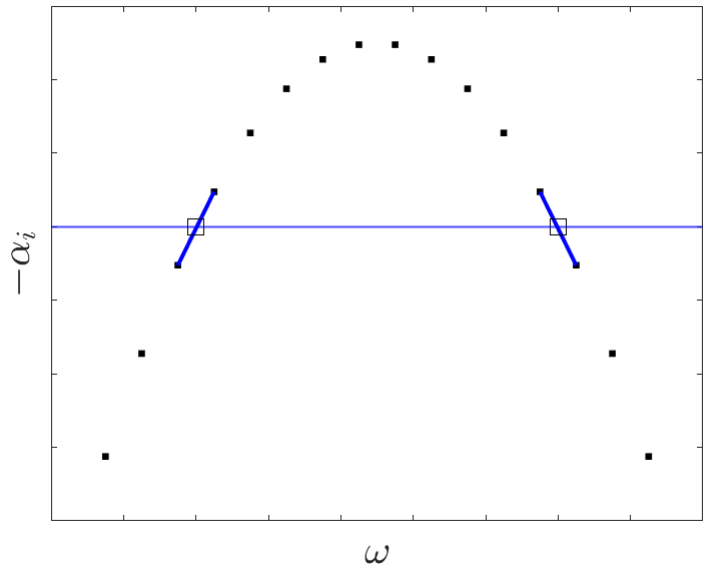

Figure 2: Schematic of the initialisation scheme.

This section is about how I go about finding initial points for an unkown new curve. It is fast and dirty but could be useful for somebody.

Initialisation

Our arclength continuation scheme requires an initial point (ω_0, R_0) such that $\alpha_i(\omega_0, \mathrm{R}_0) = 0$. To find this point on both the upper and lower branch (you can start from either and may need to do both depending on the robustness of my branchswitching rule discussed below). I have used this method the temperature dependent Crane problem but will explain how it works in the Blasius code.

- Take and initial starting Reynolds number, in our case $\mathrm{R} = 2500$.
- Define N, the number of chebychev points, from this the chebychev matrices.
- Define the mapped variables and map you base flow onto the new domain.
- Define a vector of ω over some sensible range in our case *omega = 0.01:0.01:0.1.*
- Solve $\alpha_i(\omega, \mathrm{R}_0)$ storing the dominant eigenvalue for each ω
- Find the where the imaginary eigenvalue changes sign via *sc = eig_vals(1:end-1).*eig_vals(2:end)*

- Perform linear interpolation to find ω_0 so that $\alpha_i(\omega_0, \mathrm{R}_0) = 0$.

This is performed in the script init_guess.m and the interpolation is handled by the function find_zero.

In each case $\alpha_i \sim 10^{-3}$ which is close enough to begin our scheme.

Issues?

For Cranes temperature dependent flow, the scheme sometimes picked up spurious eigenvalues and so care had to be taken to insure the correct values of ω were selected to perform the interpolation about. This code is provided in an if statement that takes the second and third location that the sign cahnges. It cannot be gaurenteed that this will always work, but it should be easily adapted for other problems where you are unsure where to start.