Thrust Normal to Velocity Vector

July 28, 2021

1 Simplifying expression for $w_p(\phi)$

The sine function is a phase shift of the cosine function, so

$$\sin(2k) = \sin(2\phi + \pi/2) = \cos(2\phi). \tag{1}$$

It is also given that

$$2\sin^2(2k) = 1 - \cos(2k). \tag{2}$$

Because $\cos(2k) = -\sin(2\phi)$ by a similar argument,

$$2\sin^2(2k) = 1 + \sin(2\phi). \tag{3}$$

Finally

$$w_p(\phi) = \underbrace{\frac{1}{\sin(2\phi + \pi/2)}}_{\cos(2\phi)}[1 \underbrace{-\frac{a_T r_0^2}{\mu}}_{-\frac{4}{\beta^2(3\pi+8)}}(3\phi + 2)] + \frac{2}{\beta^2(3\pi + 8)}\underbrace{(2\sin^2(\phi + \pi/4) + 3)}_{\sin(2\phi)+4} \tag{4}$$

$$= \frac{1}{\cos(2\phi)}\left[1 - \frac{4}{\beta^2(3\pi + 8)}(3\phi + 2)\right] + \frac{2}{\beta^2(3\pi + 8)}(\sin(2\phi) + 4) \tag{5}$$

2 Numerical evaluation of $y(\phi_0) = 0$

***Notation:* Let ϕ_0 satisfy $y(\phi_0) = 0, w_p(\phi_0) = 1$ and $\phi_{\min}$ minimize $w_p(\phi)$.**

To evaluate $y_p(\phi) = 0$, we first define `phi = (1:1:1000)*pi/4/1000`. Using this vector to formulate `y_p(Beta)`, where `Beta` is a variable constant, we have a vector of length 1000 representing $y_p(\phi)$. Note that

$$y_p(\phi) = \frac{\sqrt{\varphi(\phi)}}{\beta^2(3\pi + 8)\sin(2\phi + pi/2)}\sqrt{\frac{\mu}{r_0}}\cot(\phi + \pi/4) \tag{6}$$

$\varphi(\phi)$ was defined in Eq. 18-20 of the presentation, and the above expression comes from Eq. 21 of the presentation. This implies that $\sin(2\phi + \pi/2) \neq 0$, and further that y_p is only defined for $\varphi(\phi) \geq 0$. Thus, the vector `y_p(Beta)` must be shortened to reflect this and exclude complex numbers. From calculations, we find that the real representation of y_p is `y_p(Beta)(1:548)` using $a_T = 0.2, r_0 = 1, \mu = 1$ with $\beta \approx 1.07$. To find the section of this vector that is $\in \mathbb{R}$, we have called a function `realBreakpoint(vector)` in Figure 1. Sorting this vector and determining the corresponding ϕ_0 solves the problem, where we expect `y_minValue = 0` and `phi_0` to be the corresponding value of ϕ:

```
[y_values index_vector] = sort(y_p(Beta)(1:548));
y_minValue = y_values(1);
phi_0 = index_vector(1);
```

This approach can be extended for a changing parameter β. Since $\delta = \frac{1}{\beta^2} \in (0.001, 1)$, $\beta = \sqrt{\frac{1}{\delta}} \in (1, \sqrt{1000})$. In Octave/MATLAB, we can implement it as

```
delta = (1:1:1000)/1000;
Beta_vec = sqrt(1./delta);
```

Mathematically, we say that

$$\vec{\beta} = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} \text{ and } \vec{\delta} = \begin{bmatrix} 1/\beta_1^2 \\ \vdots \\ 1/\beta_n^2 \end{bmatrix} \tag{7}$$

where we choose $n = 1000$. Thus, by iterating through the values of `Beta_vec`, we can generate a corresponding vector `phi_m1` to represent how ϕ_0 changes with respect to δ:

```
phi_m1 = zeros(1,n);
w_1 = zeros(1,n);
w_min = zeros(1,n);
y_0 = zeros(1,n);

for i = 1:n
    y_real = y_p(Beta_vec(i))(1:realBreakpoint(y_p(Beta_vec(i))));
    [yvals idx] = sort(y_real);
    if (length(idx) > 1)
        w_1(i) = w_p(Beta_vec(i))(idx(2));
        phi_m1(i) = phi(idx(2));
        y_0(i) = yvals(2);
    endif
end
```

Figure 1: Numerically finding $\vec{\phi}_0$ over $\vec{\delta}$

In Figure 1 above, the vector `y_0` is updated with the value $y_p(\phi_{0,i})$ in each iteration for each value of β, and is expected to be 0. The vector `w_1` is updated with $w_p(\phi_{m,i})$, which is expected to be unity. As was done with `y_p(Beta)`, we have defined a vector `w_p(Beta)` to represent $w_p(\phi)$ where `Beta` is a constant that can be varied. Thus, we should expect $w_p(\vec{\phi}_{\min}) = \vec{1}$, $y_p(\vec{\phi}_0) = \vec{0} \in \mathbb{R}^n$, and the existence of $\vec{\phi}_m = \begin{bmatrix} \phi_{m,1} \\ \vdots \\ \phi_{m,n} \end{bmatrix} \in \mathbb{R}^n$ at the end of the loop. The second index of `y_p(Beta_vec(i))` is accessed to find $\phi_{0,i}$ within the loop because $\phi = 0$ always satisfies $y_p(\phi) = 0$, and we want to find the second such value. The conditional check is to account for cases where the domain of $y_p(\phi) \in \mathbb{R}$ is very small (i.e. `length(y_p(Beta_vec(i))` is 1), so we are only able to find $\phi_{0,i} = 0 \implies y(\phi_0) = 0$.

The results of ϕ_0 vs. δ are expressed in Figure 2 below. $y_p(\phi_0) \approx 0$ and $w_p(\phi_{\min}) \approx 1$ within numerical error, which verifies the validity of these results.

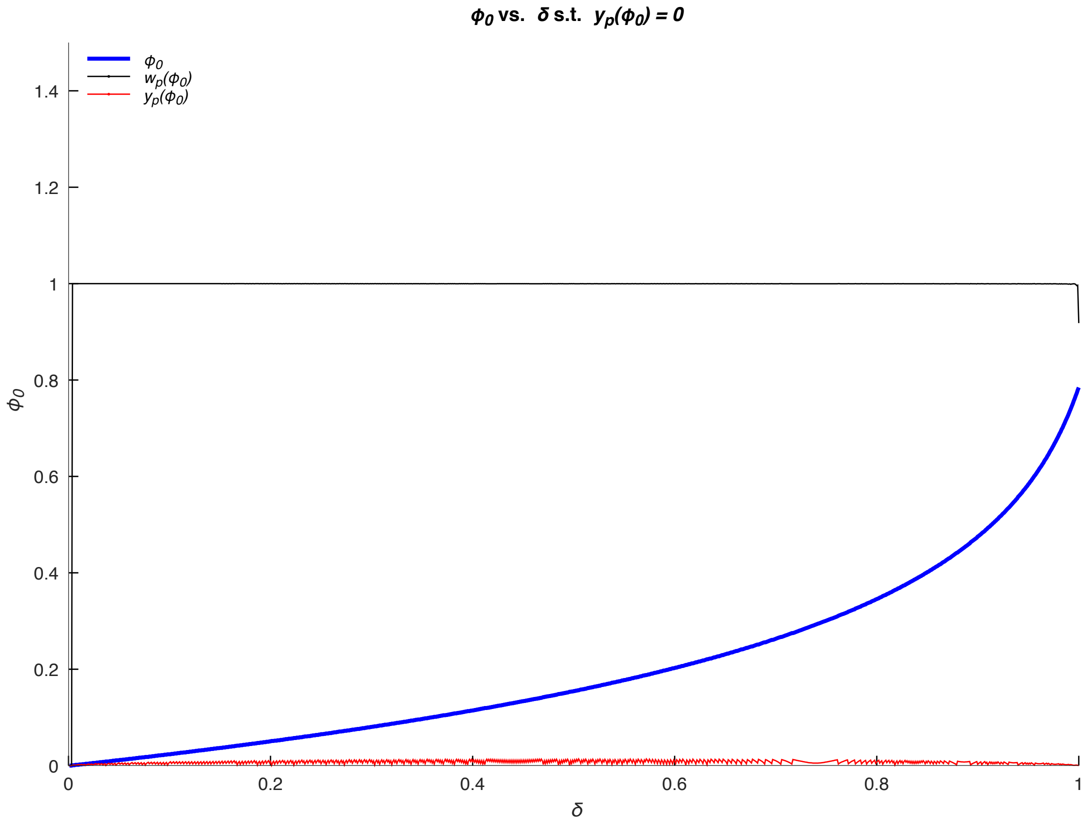

Figure 2: A plot of these results

3 Numerical minimization of $w_p(\phi)$

We use a similar approach as before to minimize $w_p(\phi)$ numerically. In Figure 1, `w_1(i)` was updated to reflect the value of $w_p(\phi_{\min,i})$ where $\phi = \phi_{\min,i}$ minimized $y_p(\phi)$ for corresponding values of β_i, δ_i from Eq. 7. The following code defines a vector `phi_m2` of length $n = 1000$ and populates it with the value of $\phi_{\min,i}$ that minimizes $w_p(\phi)$ for β_i. The corresponding minimum values are stored in another vector `w_min`. Note that the first index of the sorted vector `w_vals` is accessed, since we are looking for the absolute minimum.

```
phi_m2 = zeros(1,n);
w_min = zeros(1,n);
for i = 1:n
    [w_vals idx] = sort(w_p(Beta_vec(i)));
    phi_m2(i) = phi(idx(1));
    w_min(i) = w_vals(1);
end
```

Figure 3: Iteratively finding $\vec{\phi}_{\min}$ for various β

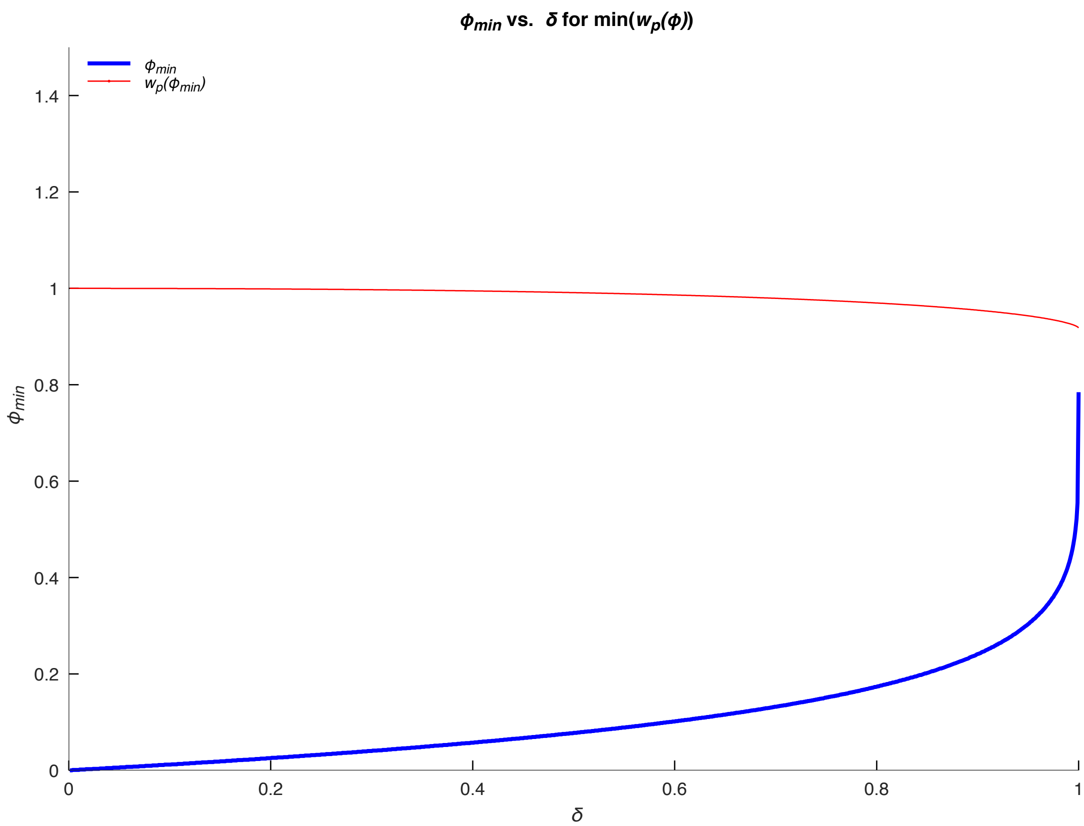

Figure 4: Values of $\phi_{\min}$ that minimize $w_p(\phi)$ for various β

Figure 4, showing $\phi_{\min}$ (representing `phi_m2`) vs. δ, summarizes these results.

4 Bisection and Newton-Raphson method for finding roots

4.1 Bisection method

We will use the bisection method to find when $y_p(\phi) = 0$ for a given tolerance ϵ. The requirement for the bisection method is that the function in question is continuous on the closed interval $[a, b]$[1], so the roots of $y_p(\phi)$ can be found. The roots of $w'(\phi)$ will be found using the Newton's method later.

4.2 Newton-Raphson method

In Eq. 40 from the presentation, $w_p{}'(\phi)$ was not expressed in terms of β, so the following expression will be used:

$$w_p{}'(\phi) = \frac{2(\sin^2(2\phi + \pi/2) - 3)\sin(2\phi + \pi/2)}{(3\pi + 8)\beta^2 \sin^3(2\phi + \pi/2)} - \frac{\cos(2\phi + \pi/2)\left(1 - \frac{4}{(3\pi+8)\beta^2}(3\phi + 2)\right)}{r_0 \sin^3(2\phi + \pi/2)} \tag{8}$$

[1] Bisection method - Wolfram MathWorld

This method cannot be used to solve for the root of $y_p(\phi)$ because although the function exists in the domain $\phi \in [0, \pi/4)$, $\lim_{\phi\to 0^+} {y_p}'(\phi)$ and $\lim_{\phi\to \pi/4^-} {y_p}'(\phi)$ do not exist, so the function is not differentiable on the entire interval[2].

From observing the graph of $y_p(\phi)$ on Page 6, this is confirmed visually. Since the endpoints of $y_p(\phi)$ on this interval are its two roots, they cannot be found through this method. The bisection method works, however, since the only requirement is continuity of $y_p(\phi)$ on the interval.

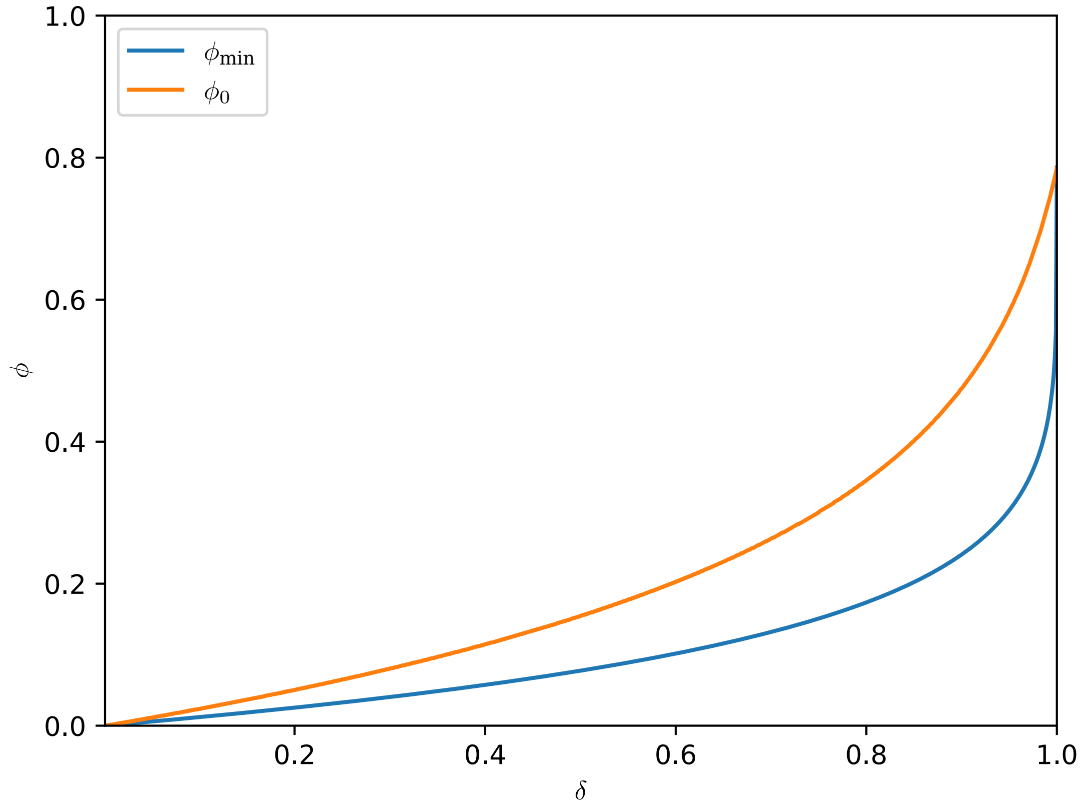

Figure 5: A plot of ϕ_{min} and ϕ_0 vs δ

The code used to implement both algorithms is located here. Figure 5 demonstrates the bisection method for finding ϕ_0 and Newton's method for ϕ_{min}.

5 Curve fitting

5.1 Objective

The least-squared method of curve fitting[3] was used to generate the closed-form equations $\phi_0(\delta)$ and $\phi_{\text{min}}(\delta)$.

The approach is to define an error function $E(\phi) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\phi_i - \phi(\delta_i))^2}$ that represents the RMS error between the fitted curve and original data. n is the number of data points in the set

[2] Newton's method

[3] Curve fitting: least squares methods

$\{(\delta_1, \phi_1), \ldots, (\delta_n, \phi_n)\}$ and $\phi(\delta)$ represents the fitted curve. Because a polynomial curve is desired, the coefficients of the terms in $\delta(\phi)$ belong to $\vec{c} \in \mathbb{R}^{k+1}$, so there are $k+1$ coefficients in the fitted polynomial. Since $\phi(\delta; \vec{c})$, $E : \mathbb{R}^{k+1} \to \mathbb{R}; \vec{c}$. The objective is then to find a $\vec{c}$ which minimizes $E(\vec{c})$. Similarly, each $\phi(\delta_i)$ becomes a function of $\vec{c}$.

Minimizing $E(\vec{c}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\phi_i - \phi(\delta_i))^2}$ is the same as minimizing $E_1(\vec{c}) = \sum_{i=1}^{n}(\phi_i - \phi(\delta_i))^2$, so the next step is to solve

$$\frac{\partial E_1}{\partial c_j} = 2\sum_{i=1}^{n}(\phi_i - \phi(\delta_i))\frac{\partial \phi(\delta_i)}{\partial c_j} = 0 \tag{9}$$

$$\sum_{i=1}^{n}\frac{\partial \phi(\delta_i)}{\partial c_j}\phi_i = \sum_{i=1}^{n}\frac{\partial \phi(\delta_i)}{\partial c_j}\phi(\delta_i) \tag{10}$$

$$\sum_{i=1}^{n}\delta_i^{j-1}\phi_i = \sum_{i=1}^{n} c_j\delta_i^{2j-1} + \ldots + c_1\delta_i^{j-1} = \sum_{\ell=1}^{j}\sum_{i=1}^{n} c_{j-\ell+1}\delta_i^{2j-\ell-1} \tag{11}$$

where $j = 1, \ldots, k+1$ and the polynomial is of degree k.

A vectorized implementation yields

$$\begin{bmatrix} \sum_{i=1}^{n}\delta_i^k\phi_i \\ \vdots \\ \sum_{i=1}^{n}\delta_i^1\phi_i \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n}\delta_i^{2k} & \cdots & \sum_{i=1}^{n}\delta_i^k \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{n}\delta_i^k & \cdots & \sum_{i=1}^{n} \end{bmatrix} \begin{bmatrix} c_{k+1} \\ \vdots \\ c_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} c_{k+1}\delta_i^{2k} & \cdots & \sum_{i=1}^{n} c_1\delta_i^k \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} c_{k+1}\delta_i^k & \cdots & \sum_{i=1}^{n} c_1 \end{bmatrix} \tag{12}$$

$$\vec{b} = A\vec{c} \tag{13}$$

The matrix-vector equation $A\vec{c} = \vec{b}$ can then be solved for $\vec{c}$ using an optimized linear algebra library, and the resulting coefficients used to generate a function $\phi(\delta)$.

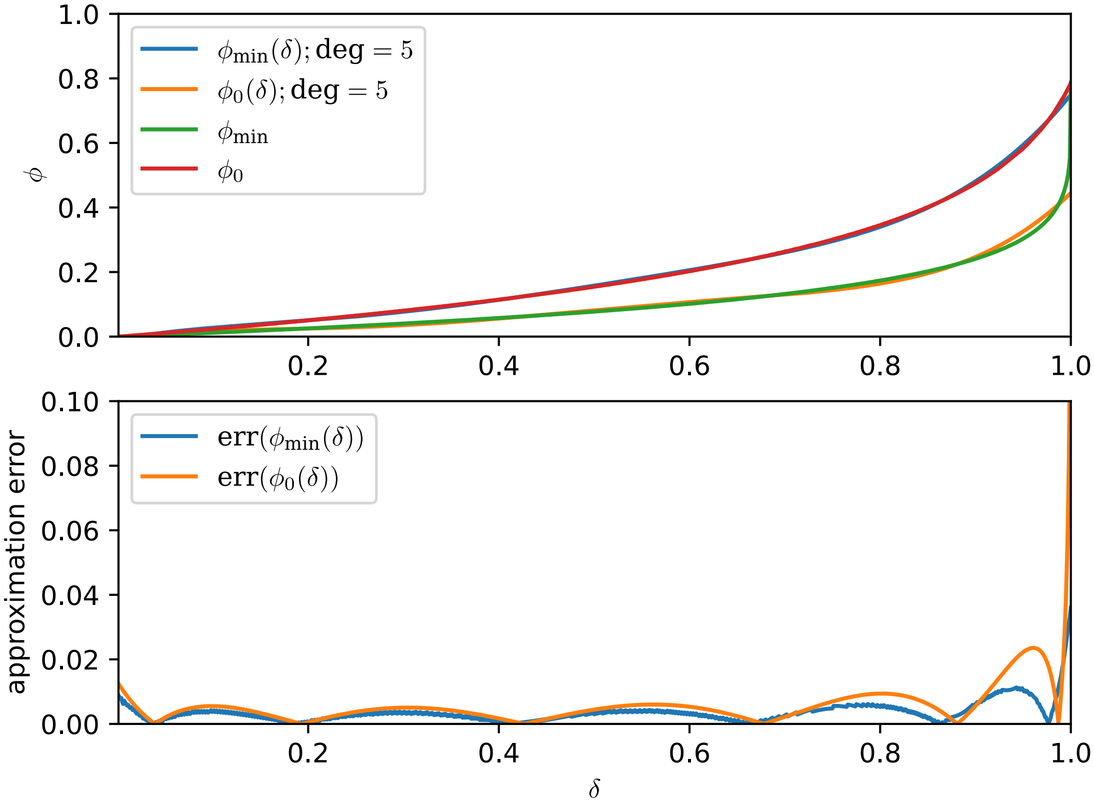

Figure 6: Using degree 5 polynomials to approximate $\phi_0(\delta)$ and $\phi_{\min}(\delta)$

A better approximation is desired since even with degree 5, a polynomial approximation cannot fit well to the smooth curves of ϕ_0 and $\phi_{\min}$. This is evidenced by the Runge effect visible in the error plot of Figure 6.

5.2 Attempted approximation using Chebyshev polynomial interpolation

The Chebyshev polynomial approximations resulted in a more pronounced Runge effect, as depicted below.

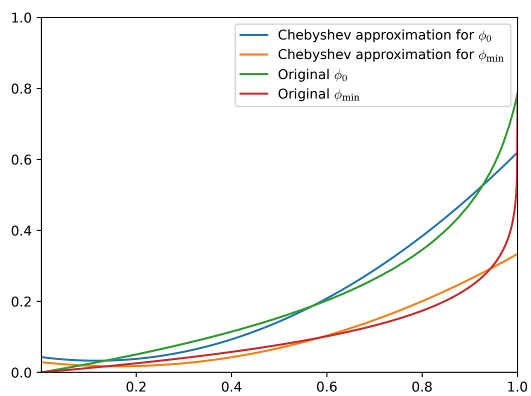

Figure 7: Chebyshev polynomial approximations

5.3 Attempted approximation using exponential function

By far the most accurate fit before the original polynomial approximation, an exponential function of the form $\phi = a\exp(b(\delta - c)) + d$ was used. Significant Runge effect is still evident in the plot below.

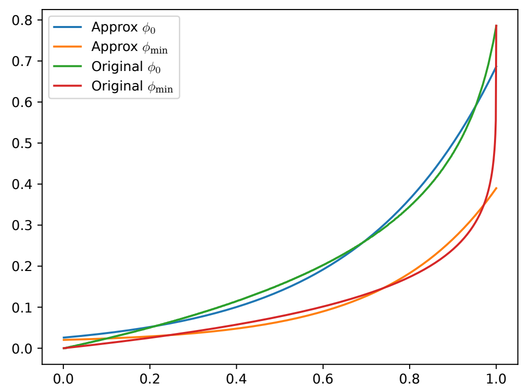

Figure 8: Exponential approximations

5.4 Attempted elliptical integral of first kind approximation

Through parameterizing the first-order elliptic integral $K(k;n)$ for some $n \in \mathbb{R}$ and $k \in [0,1]$ for both translation and dilation control, curve fitting was attempted:

$$K(k;n) = -\frac{\pi}{2n} + \int_0^1 \frac{dx}{n\sqrt{(1-x^2)(1-k^2x^2)}} \tag{14}$$

The optimization routine returned a value of $n = 1$ for both ϕ_0 and $\phi_{\min}$, indicating that the first-order approximation is not effective. The plot is shown below.

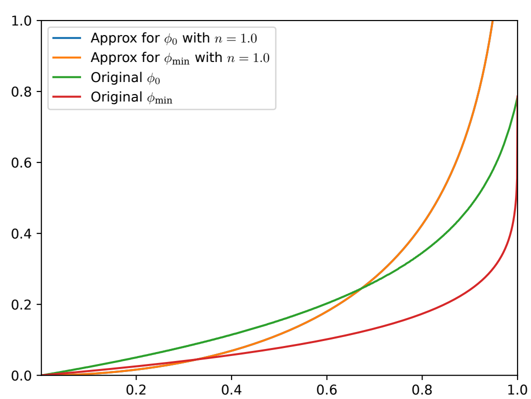

Figure 9: Elliptical approximations

Further parameterization attempts of $K(k)$ did not yield better results, as the curvature of the elliptic integral function cannot be manipulated easily to fit the data.

5.5 Univariate spline approximation

This method yielded the best results. A univariate spline of degree $k = 3$ with smoothness factor $s = 0.001$ was used to approximate ϕ_0 and $\phi_{\min}$. Figure 10 shows that the method was successful, with minimized Runge effect and typical error magnitudes lower than that of the polynomial approximation.

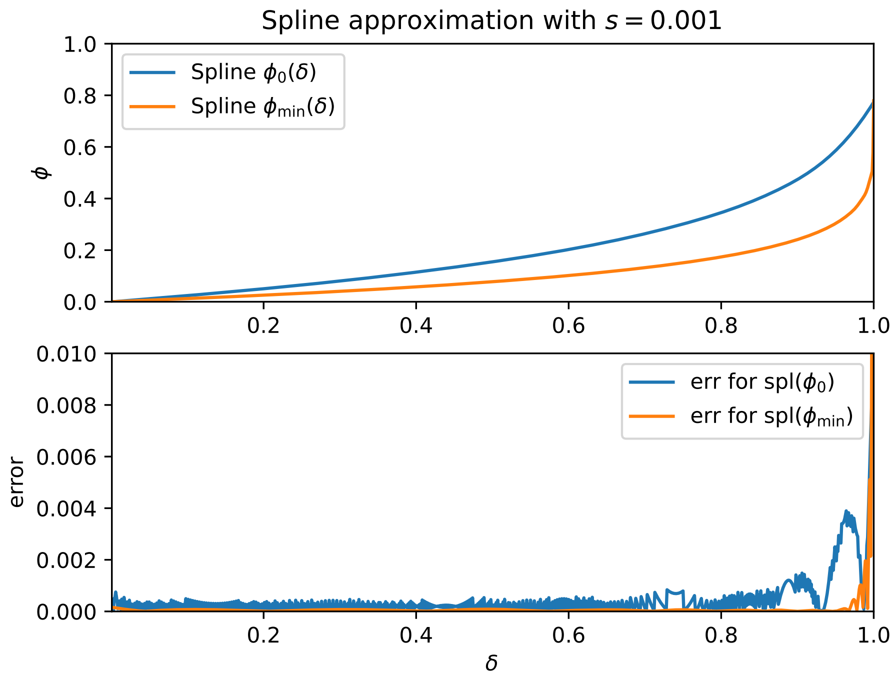

Figure 10: Spline approximation plot

The computer output below details the properties of each spline:

```
    ---------- phi_0 spline approx ----------
LSQ Error =         0.001000051241244917
Knots =         [0.001 0.501 0.751 0.876 1.    ]
Coeffs =        [-4.40529375e-04  3.92784349e-02  1.04654145e-01  2.56034241e-01
  4.15005836e-01  5.76096213e-01  7.73738936e-01]

---------- phi_min spline approx ----------
LSQ Error =         0.0009999540547108812
Knots =         [0.001 0.501 0.626 0.751 0.814 0.876 0.907 0.938 0.969 0.977 0.985 0.993
 0.997 1.   ]
Coeffs =        [-4.58365176e-05  1.99513345e-02  4.68542634e-02  1.06810792e-01
  1.41001801e-01  1.78919227e-01  2.10975405e-01  2.46198594e-01
  2.81539962e-01  3.21468750e-01  3.64172354e-01  3.99187610e-01
  4.27132615e-01  5.02562701e-01  4.92718655e-01  7.78749733e-01]
```

6 Simplifying the expression $1 - w_p(\phi)^2$ from $y_p(\phi)$

Objective: obtain the simplest expression for $1 - w_p(\phi)^2$ in terms of ϕ, β.

The original expression for $1 - w_p(\phi)^2$ is below, with the denominator removed for simplicity.

$$
\begin{aligned}
(3\pi+8)^2\beta^4\sin^2(2k)(1-w_p(\phi)^2) &= \underbrace{(3\pi+8)^2\beta^4\sin^2(2k) - \left[(3\pi+8)^2\beta^4 - 8(3\phi+2)(3\pi+8)\beta^2 + 16(3\phi+2)^2\right]}_{\gamma_1(\phi)} && (15)\\
&- \underbrace{2\sin(2k)(4\sin^2(k)+6)((3\pi+8)\beta^2 - 4(3\phi+2))}_{\gamma_2(\phi)} && (16)\\
&- \underbrace{\sin^2(2k)(16\sin^4(k) + 48\sin^2(k) + 36)}_{\gamma_3(\phi)} && (17)\\
&= \gamma_1(\phi) - \gamma_2(\phi) - \gamma_3(\phi) && (18)
\end{aligned}
$$

Applying Equations 1-3 to the denoted sub-functions $\gamma_1(\phi), \gamma_2(\phi), \gamma_3(\phi)$, we get

$$
\begin{aligned}
\gamma_1(\phi) &= (3\pi+8)^2\beta^4\left(\cos^2(2\phi)-1\right) + 8(3\phi+2)\left((3\pi+8)\beta^2 - 2(3\phi+2)\right) && (19)\\
\gamma_2(\phi) &= -2(8\cos(2\phi) + 2\sin(2\phi)\cos(2\phi))\left((3\pi+8)\beta^2 - 4(3\phi+2)\right) && (20)\\
&= -4\cos(2\phi)(\sin(2\phi)+2)\left((3\pi+8)\beta^2 - 4(3\phi+2)\right) && (21)\\
\gamma_3(\phi) &= -\cos^2(2\phi)\left(16\sin\left(\phi+\frac{\pi}{4}\right)^4 + 48\sin^2\left(\phi+\frac{\pi}{4}\right) + 36\right) && (22)\\
&= -\cos^2(2\phi)\left(4\sin^2(2\phi)+6\right)^2 && (23)
\end{aligned}
$$

Putting this together, a simplified expression is

$$
\begin{aligned}
1 - w_p(\phi)^2 &= \frac{\gamma_1}{(3\pi+8)^2\beta^4\cos^2(2\phi)} + \frac{\gamma_2}{(3\pi+8)^2\beta^4\cos^2(2\phi)} + \frac{\gamma_3}{(3\pi+8)^2\beta^4\cos^2(2\phi)} && (24)\\
&= \underbrace{\frac{\cos^2(2\phi)-1}{\cos^2(2\phi)}}_{\frac{-(1-\cos^2(2\phi))}{\cos^2(2\phi)} = \frac{-\sin^2(2\phi)}{\cos^2(2\phi)} = \boxed{-\tan^2(2\phi)}} + \frac{8(3\phi+2)((3\pi+8)\beta^2 - 2(3\phi+2))}{(3\pi+8)^2\beta^4\cos^2(2\phi)} && (25)\\
&- \frac{4(\sin(2\phi)+2)((3\pi+8)\beta^2 - 4(3\phi+2))}{(3\pi+8)^2\beta^4\cos(2\phi)} - \frac{\left(4\sin^2(2\phi)+6\right)^2}{(3\pi+8)^2\beta^4} && (26)
\end{aligned}
$$

Because $y_p(\phi) = \sqrt{1 - w_p(\phi)^2}\sqrt{\frac{\mu}{r_0}}\cot(\phi + \pi/4)$, by substitution

$$
\boxed{y_p(\phi) = \sqrt{-\tan^2(2\phi) + \frac{8(3\phi+2)((3\pi+8)\beta^2 - 2(3\phi+2))}{(3\pi+8)^2\beta^4\cos^2(2\phi)} - \frac{4(\sin(2\phi)+2)((3\pi+8)\beta^2 - 4(3\phi+2))}{(3\pi+8)^2\beta^4\cos(2\phi)} - \frac{(4\sin^2(2\phi)+6)^2}{(3\pi+8)^2\beta^4}}\sqrt{\frac{\mu}{r_0}}\cot(\phi+\pi/4)} \quad (27)
$$

7 Chebyshev and Lagrange interpolation

In the following functions, the domains are defined for all $\delta > 0$ or $r > 0$, so in finding Chebyshev nodes for interpolation, only nodes in this domain are filtered from a list of possibilities. Thus, when selecting n nodes, we actually select more, of which the negative values are discarded. Using the Lagrange polynomial interpolation method, these nodes are used to find an approximating polynomial.

7.1 Interpolation of $\phi_0, \phi_{\min}$

In Figure 11 and 12, 9 Chebyshev nodes were used to generate an approximating polynomial.

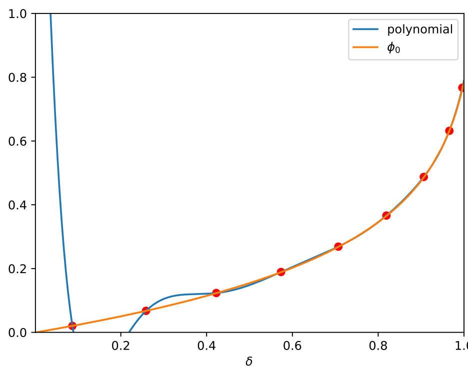

Figure 11: Lagrange polynomial approximation of ϕ_0 with equation g

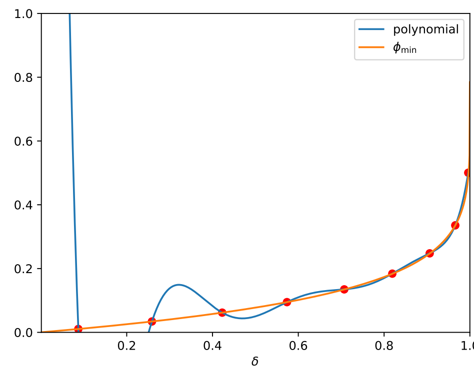

Figure 12: Lagrange polynomial approximation of $\phi_{\min}$

Polynomial for ϕ_0:

$$f_1(\delta) = 1044\delta^8 - 4892\delta^7 + 9693\delta^6 - 10531\delta^5 + 6800\delta^4 - 2638\delta^3 + 589\delta^2 - 67\delta^1 + 3 \tag{28}$$

Polynomial for $\phi_{\min}$:

$$f_2(\delta) = 3968\delta^8 - 18749\delta^7 + 37405\delta^6 - 40874\delta^5 + 26519\delta^4 - 10327\delta^3 + 2311\delta^2 - 263\delta^1 + 11 \tag{29}$$

The coefficients displayed here are rounded to the nearest whole number in order to make for readable output.

7.2 Interpolation of $y(r)$

As shown in Figure 13, the Cheybshev node method yielded far more accurate results for the function $y(r)$, save for the initial spike/irregularity in the function when it is increasing.

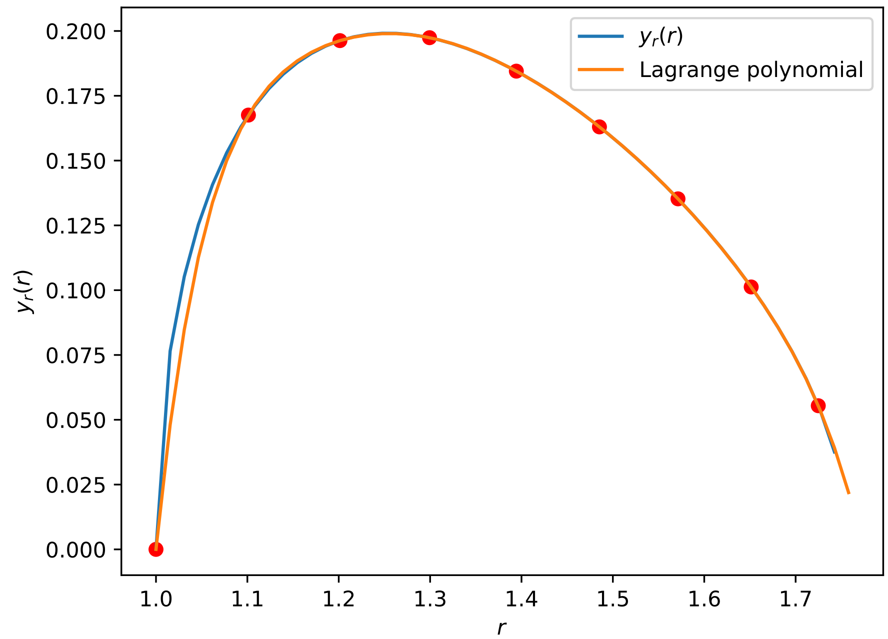

Figure 13: Lagrange polynomial approximation for $y(r)$

The equation of the Lagrange polynomial is:

$$f_3(r) = -282r^8 + 3195r^7 - 15828r^6 + 44706r^5 - 78750r^4 + 88591r^3 - 62158r^2 + 24872r^1 - 4346 \quad (30)$$

7.2.1 Integral approximation for $\frac{dr}{dt} = y(r)$

The definition given is $y(r) = \frac{dr}{dt}$. This can be rearranged to

$$\frac{dr}{y(r)} = dt \quad (31)$$

Taking the indefinite integral on both sides (the limit on the side with r is from $r_0 = 1$ to r_m where $y(r_m) = 1 | r_m > 1$ and from 0 to some t_f for the side with t) we get

$$\int_1^{r_m} \frac{dr}{y(r)} = \int_0^{t_f} dt \quad (32)$$

Using the approximation in Eq. 30, this becomes

$$\int_1^{r_m} \frac{dr}{f_3(r)} = \int_0^{t_f} dt \quad (33)$$

Using an optimized solver, the unknown is $t_f = 4.52624262$.

7.3 Limitations and future goals

- The NumPy Lagrange solver does not allow for specification of some tolerance ϵ
  - A potential solution could be to implement the Lagrange polynomial method from scratch, where a tolerance could be specified
- Another interpolation routine that could be used is the Newton polynomial method, which could also be implemented manually in order to specify a tolerance