Pseudo-arclength Continuation

Anonymous and Sarah Brockman

anonymous@umass.edu, sbrockman@umass.edu

Abstract

Pseudo-arclength continuation is a continuation scheme for finding solutions of an equation or systems of equations that depend on a parameter α, such as $F(x, \alpha) = 0$. Standard methods for approximating solutions, such as Newton's method, break down at turning point bifurcations or branching points because the Jacobian matrix becomes singular. Thus, there is a need for more advanced continuation schemes such as pseudo-arclength continuation to continue finding solutions past such a point.

1. Standard Solution Methods

Newton's method, also called the Newton-Raphson method, is a method of finding solutions to an equation or system of equations of the form $F(x) = 0$, or in our case $F(x, \alpha) = 0$ for parameter α. Newton's method for a single equation takes the form: $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$. Newton's method for systems takes the form: $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \boldsymbol{J}^{-1}(\boldsymbol{x}_i)F(\boldsymbol{x}_i)$ where $\boldsymbol{J}$ is the Jacobian matrix. Clearly, if the Jacobian becomes singular, Newton's method will break down. When we reach a turning point bifurcation or branching point, this will happen, and we cannot use Newton's method near that point. We need some way of continuing past that point so we can continue using Newton's method.

Preprint submitted to Math 552

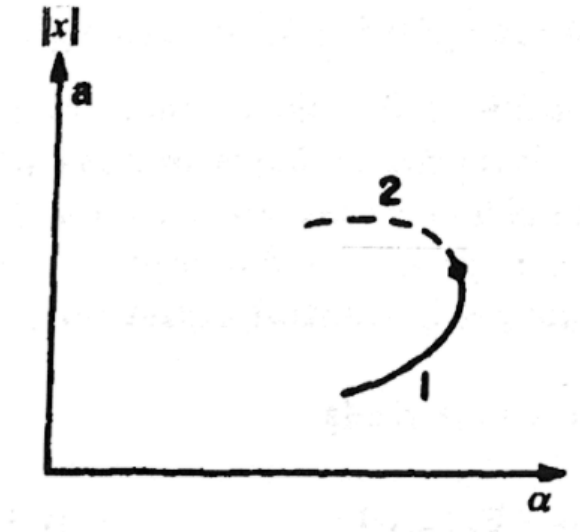

Figure 1: Example of a Turning point bifurcation

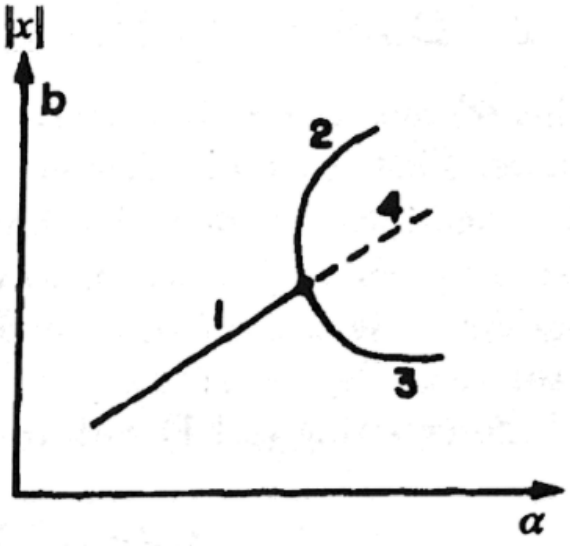

Figure 2: Example of a branching point bifurcation

2. Background

The pseudo-arclength continuation scheme solves the problems mentioned above. The arclength s is used as a continuation parameter, so $\boldsymbol{x}$ and $\boldsymbol{\alpha}$ are taken to be functions of s. So, now we need to find $\boldsymbol{x}$ and $\boldsymbol{\alpha}$ such that $F(\boldsymbol{x}(s), \boldsymbol{\alpha}(s)) = 0$. The pseudo-arclength continuation scheme is called a predictor-corrector method. When a bifurcation point is found, a tangent vector is found at a point slightly before the bifurcation point. This tangent vector can be used to predict the next values of $\boldsymbol{x}$ and $\boldsymbol{\alpha}$. Since the tangent predictor is usually not quite accurate, a corrector step is usually needed. For this, we can use the Newton-Raphson scheme mentioned above. We perform iterations of Newton's method orthogonal to the tangent vector. Figure 3 illustrates the behavior of the method well.

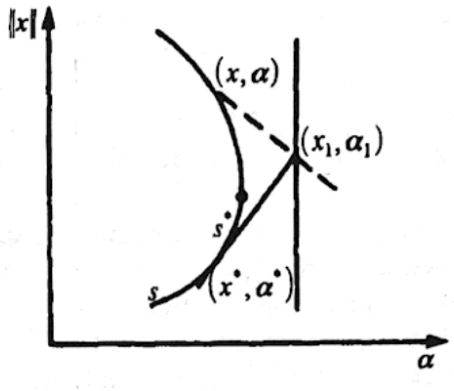

Figure 3: Pseudo-arclength continuation

3. Implementation

We first implemented this method on the following equation:

$$f(x) = x - \alpha \frac{19}{4} \frac{e^x}{1 + \alpha e^x}, \tag{1}$$

which depends on the parameter α. When we attempt to find solutions x of this equation using Newton's method, we obtain the results seen in figure 4. Clearly, Newton's method encounters a problem around $\alpha = 0.1$, when it reaches a bifurcation point. However, we can use the pseudo-arclength

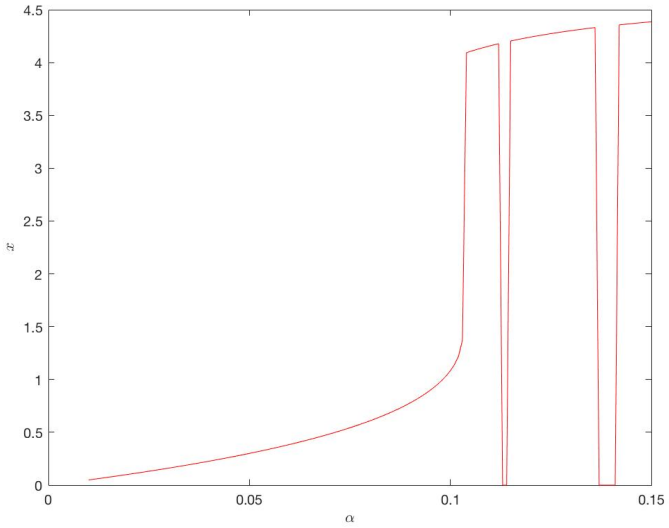

Figure 4: Newton's method for Equation (1)

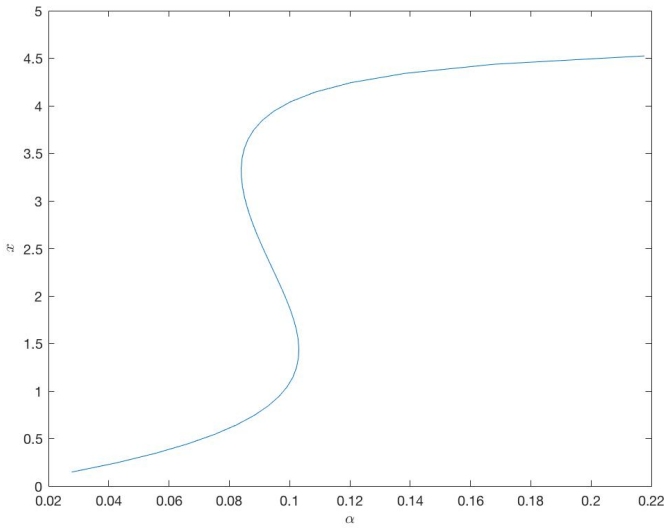

Figure 5: Pseudo-arclength method for Equation (1)

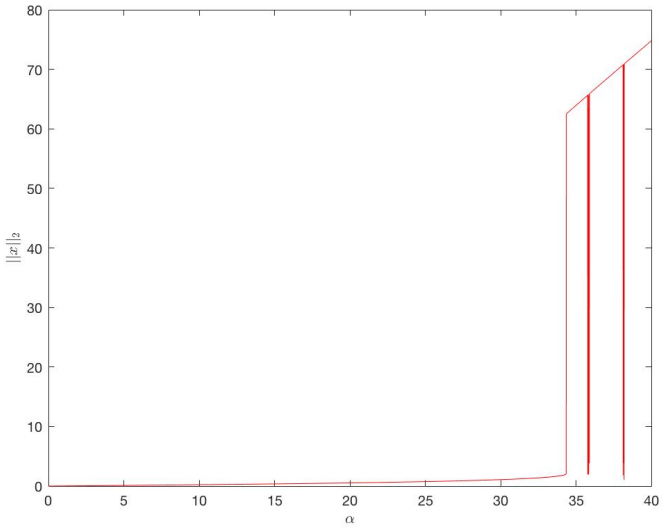

Figure 6: Newton's method for Equation (2)

method to continue past this point and produce a smooth solution curve, as seen in figure 5.

We next applied the pseudo-arclength method to the following system of equations, which model a two-component enzyme system:

$$\begin{aligned} \frac{dx_1}{dt} &= (\alpha - x_1) + (x_2 - x_1) - \rho R(x_1), \\ \frac{dx_2}{dt} &= (\alpha + \mu - x_2) + (x_1 - x_2) - \rho R(x_2), \end{aligned} \tag{2}$$

where

$$R(x) = \frac{x}{1 + x + \kappa x^2}.$$

We fixed the parameters $\mu = 0$, $\rho = 100$, and $\kappa = 1$ and attempted to find solutions $x = (x_1, x_2)$ while varying the parameter α. Like with Equation (1), Newton's method reaches a bifurcation point when attempting to solve this system (figure 6). Applying the pseudo-arclength method allows the solution to once again bypass this bifurcation and obtain a smooth solution curve. These results can be seen in figure 7.

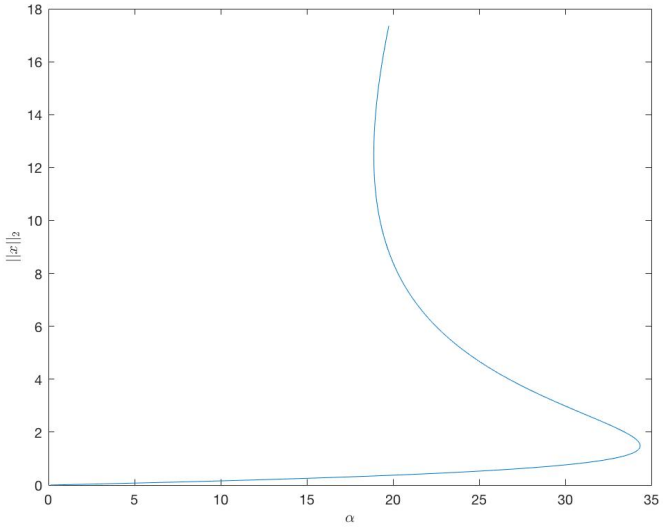

Figure 7: Pseudo-arclength method for Equation (2)

```
function x = newton(x, f, fprime)
tol = 1e-12;
max_iterations = 1000;
i = 0;
while (abs(f(x) - 0) > tol) && i < max_iterations
    x = x - f(x) / fprime(x);
    i = i+1;
end
```

Figure 8: Newton's method for a single equation

```
clearvars; close all;

% parameter range
a_vals = 0.01:0.001:0.15;
a_range = length(a_vals);
x_vals = zeros(length(a_vals),1);

for i = 1:a_range
    a = a_vals(i);
    f = @(x) x - a * 19/4 * exp(x) / (1 + a * exp(x));
    fprime = @(x) 1 - a * 19/4 * (exp(x)*(1+a*exp(x)) - exp(x)*a*exp(x))/((1+a*exp(x))^2);

    % Newton's method
    x = newton(0.05, f, fprime);
    x_vals(i) = x;
end
figure;
plot(a_vals, x_vals,'-r');
xlabel('$\alpha$','interpreter','latex');
ylabel('$x$','interpreter','latex');
```

Figure 9: Newton's method driver for Equation (1)

```
clearvars; clc; clear; format long;

% define function f
f = @(x,a) x - a * 19/4 * exp(x) / (1 + a * exp(x));
fx = @(x,a) 1 - a * 19/4 * (exp(x)*(1+a*exp(x)) - exp(x)*a*exp(x))/((1+a*exp(x))^2);
fa = @(x,a) -19 * exp(x) / (4 * (exp(x) * a + 1)^2);

g = @(x,a,xstar,astar,xstarp,astarp,ds) ...
     transpose(x-xstar) * xstarp + ( a - astar ) * astarp - ds;

xstar = 0.049385858605231; % root
astar = 0.01; % parameter for root
dir = + 1; % direction
ds = 1e-1; % step size along arc length
r = 1; % relaxation parameter for Newton's method
tol = 10e-10; % tolerance
numIterations = 45;

for i = 1:numIterations
    Jx = fx(xstar, astar);
    Ja = fa(xstar, astar);

    z = -Ja / Jx;

    % determine direction
    if i == 1
        astarp = dir / sqrt(1 + transpose(z) * z);
    else
        check_dir = transpose(xstore) * z + astore;
        if check_dir > 0
            dir = 1;
        else
            dir = -1;
        end
        astarp = dir / sqrt(1 + transpose(z) * z);
    end
```

Figure 10: Pseudo-arclength method for Equation (1), part 1

```
    xstarp = astarp * z;
    astore = astarp;
    xstore = xstarp;

    xc = xstar + xstarp * ds;
    ac = astar + astarp * ds;

    % arbitrary initial values to start loop
    xp = xc + 2 * tol;
    ap = ac + 2 * tol;
    while abs(xc - xp) > tol || abs(ac - ap) > tol
        xp = xc;
        ap = ac;

        z2 = -fa(xp,ap) / fx(xp,ap);
        z1 = -f(xp,ap) / fx(xp,ap);

        da = -(g(xp,ap,xstar,astar,xstarp,astarp,ds)+transpose(xstarp)*z1)...
            /(astarp+transpose(xstarp)*z2);
        dx = z1 + z2 * da;

        xc = xp + r * dx;
        ac = ap + r * da;
    end

    xstar = xc;
    astar = ac;

    % store results for plotting
    xsto(i) = xstar;
    asto(i) = astar;
end
plot(asto,xsto);
xlabel('$\alpha$','interpreter','latex');
ylabel('$x$','interpreter','latex');
```

Figure 11: Pseudo-arclength method for Equation (1), part 2

```
function x = newton_sys(x, f, fprime)
tol = 1e-12;
max_iterations = 1000;
i = 0;
while (abs(norm(f(x)) - 0) > tol) && i < max_iterations
    x = x - inv(fprime(x)) * f(x);
    i = i+1;
end
```

Figure 12: Newton's method for a system of equations

```
clearvars; close all;

% parameter range
a_vals = 0.01:0.01:40;
a_range = length(a_vals);
x_vals = zeros(length(a_vals),1);

m = 0;
r = 100;
k = 1;
R = @(x) x / (1 + x + k * x^2);
Rp = @(x) ((1 + x + k * x^2) - x * (1 + k * 2 * x)) / ((1 + x + k * x^2)^2);

for i = 1:a_range
    a = a_vals(i);
    f = @(x) [(a - x(1)) + (x(2) - x(1)) - r * R(x(1));...
        (a + m - x(2)) + (x(1) - x(2)) - r * R(x(2))];
    fprime = @(x) [-2 - r * Rp(x(1)), 1; 1, -2 - r * Rp(x(2))];

    % Newton's method
    x = newton_sys([0,0], f, fprime);
    x_vals(i) = norm(x);
end
figure;
plot(a_vals, x_vals,'-r');
xlabel('$\alpha$','interpreter','latex');
ylabel('$||x||_{2}$','interpreter','latex');
```

Figure 13: Newton's method driver for Equation (2)

```
clearvars; clc; clear; format long;

% define function f
m = 0;
r = 100;
k = 1;
R = @(x) x / (1 + x + k * x^2);
Rp = @(x) ((1 + x + k * x^2) - x * (1 + k * 2 * x)) / ((1 + x + k * x^2)^2);
f = @(x,a) [(a - x(1)) + (x(2) - x(1)) - r * R(x(1));...
    (a + m - x(2)) + (x(1) - x(2)) - r * R(x(2))];
fx = @(x,a) [-2 - r * Rp(x(1)), 1; 1, -2 - r * Rp(x(2))];
fa = @(x,a) [1;1];

g = @(x,a,xstar,astar,xstarp,astarp,ds) ...
     transpose(x-xstar) * xstarp + ( a - astar ) * astarp - ds;

xstar = [0.01;0.01]; % root
astar = 0.01; % parameter for root
dir = + 1; % direction
ds = 1e-1; % step size along arc length
r = 1; % relaxation paramter for Newton's method
tol = 10e-10; % tolerance
numIterations = 600;

for i = 1:numIterations
    Jx = fx(xstar, astar);
    Ja = fa(xstar, astar);

    z = -Jx \ Ja;
```

```
    % determine direction
    if i == 1
        astarp = dir / sqrt(1 + transpose(z) * z);
    else
        check_dir = transpose(xstore) * z + astore;
        if check_dir > 0
            dir = 1;
        else
            dir = -1;
        end
        astarp = dir / sqrt(1 + transpose(z) * z);
    end

    xstarp = astarp * z;
    astore = astarp;
    xstore = xstarp;

    xc = xstar + xstarp * ds;
    ac = astar + astarp * ds;
```

Figure 14: Pseudo-arclength method for Equation (2), part 1

```
    % arbitrary initial values to start loop
    xp = xc + 2 * tol;
    ap = ac + 2 * tol;
    while norm(xc - xp) > tol || norm(ac - ap) > tol
        xp = xc;
        ap = ac;

        z2 = -fx(xp,ap) \ fa(xp,ap);
        z1 = -fx(xp,ap) \ f(xp,ap);

        da = -(g(xp,ap,xstar,astar,xstarp,astarp,ds)+transpose(xstarp)*z1)...
            /(astarp+transpose(xstarp)*z2);
        dx = z1 + z2 * da;

        xc = xp + r * dx;
        ac = ap + r * da;
    end

    xstar = xc;
    astar = ac;

    % store results for plotting
    xsto(i) = norm(xstar);
    asto(i) = astar;
end
plot(asto,xsto);
xlabel('$\alpha$','interpreter','latex');
ylabel('$||x||_{2}$','interpreter','latex');
```

Figure 15: Pseudo-arclength method for Equation (2), part 2