The Continuation Method for Algebraic Nonlinear Equations

John Burkardt
Department of Scientific Computing
Florida State University

.....

https://people.sc.fsu.edu/~jburkardt/presentations/continuation_2014_fsu.pdf

June 27, 2022

Abstract

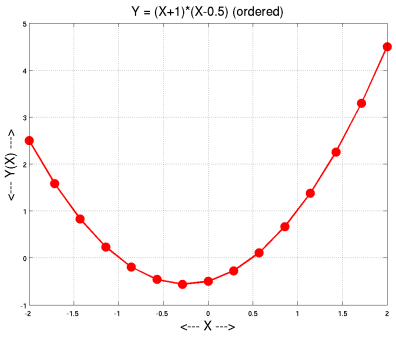
The continuation method produces a sequence of solutions to a set of algebraic nonlinear equations with one degree of freedom. The computed points indicate the shape of an underlying smooth solution curve. An efficient method should know how to handle gradual changes in direction of the curve. As an illustration, we compute points on the unit circle.

1 Simple Continuation: Plotting Y=F(X)

Continuation is a method used to compute a sequence of closely-spaced solutions to a given set of one or more mathematical equations. In the most common case, we can think of these as geometric points that lie on a curve.

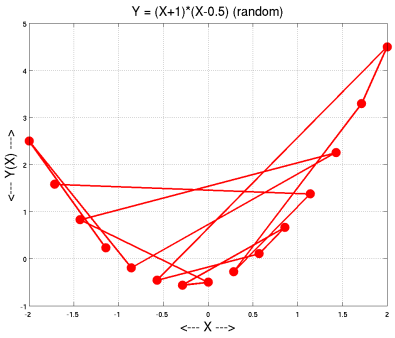
In the simple case where there is one mathematical equation, and it can be written as an explicit formula, we can easily make a sketch of the curve of solution points without requiring any special techniques. Thus, neighboring points on a parabola can be plotted in MATLAB by

```
x = linspace ( -2.0, +2.0, 15 );
y = ( x + 1.0 ) .* ( x - 0.5 );
plot ( x, y, 'r.-' );
```



You might not realize how important it is that we create the sample points in order, so that we can connect neighbors. Observe the disaster that occurs when we randomly reorder the x values:

```
x = linspace ( -2.0, +2.0, 15 );
i = randperm ( 15 );
x = x(i);
y = ( x + 1.0 ) .* ( x - 0.5 );
plot ( x, y, 'r.-' );
```

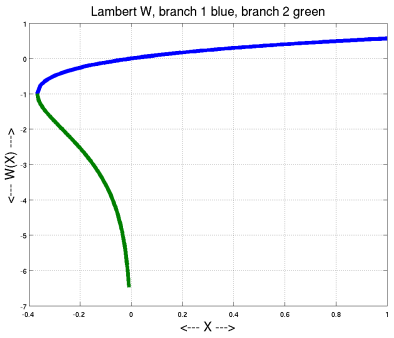


The point we make here is that, to create a useful sampling of a curve, it is necessary both to be able to determine points on the curve, **and** to be able to arrange those points in some orderly way that records which points are neighbors.

One reason the parabola problem is so easy is that we are given an explicit formula. I am then free to pick any value of x and can easily produce the corresponding y. But there are many problems which can be answered by producing a similar plot of points, but which are much harder, for various reasons. One issue arises when the function is implicit. Consider the Lambert function $y = W(x)$, defined implicitly by

$$x = W(x)e^{W(x)}$$

Do any solutions exist? Do the solutions form a curve? Is the curve closed, or does y or x go to infinity? If I find one solution, does that help me determine others?



We will see that the method of continuation can be used in such difficult situations, providing a way to produce an orderly sequence of solutions to an implicit equation.

2 Going in Circles

As a simple example of the problems that can be handled by continuation, let's suppose we are given the equation for the points on the unit circle, and suppose we actually have no idea about the geometry of the

solution points.

In other words, we are trying to figure out if there are solutions to

$$x^2 + y^2 = 1$$

If we wish to think in terms of Newton's method, we would think of this problem as

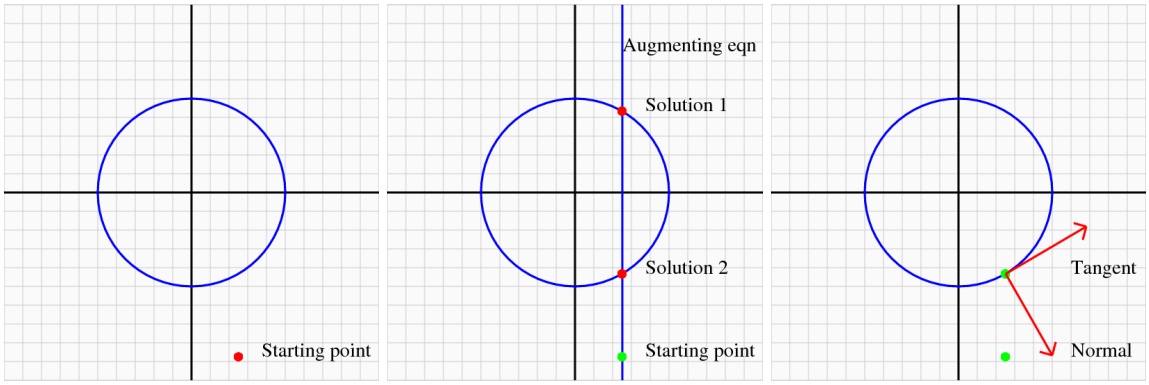
$$f(x, y) = x^2 + y^2 - 1$$

and state that we are searching for solutions to $f(x, y) = 0$.

Newton's method can't be used immediately, because we have two variables, but only one equation. But if we really want to use Newton's method, we can do so by coming up with a second, sensible augmenting equation. For example, we might ask, are there solutions to this equation if we assume that the x parameter is equal to $\frac{1}{2}$? In that case, our system of equations becomes

$$\begin{aligned} x^2 + y^2 &= 1 \\ x &= \frac{1}{2} \end{aligned}$$

and Newton's method can be applied, returning the solution $(x, y) = (\frac{1}{2}, \frac{\sqrt{3}}{2})$ or the corresponding solution with the y value negated.



We were, of course, lucky to investigate the value $x = \frac{1}{2}$. Depending on the value we choose for x, this particular system has 0, 1 or 2 equations.

Now, although we have computed a very particular solution to the system, we don't have any idea of the shape of the curve formed by all the solutions. Can we somehow use this first solution point as a key to obtaining an outline of the whole set of solutions?

It turns out that, if we have a point (x, y) that lies on the circle, then there are two very interesting direction vectors at that point:

- $\vec{n}$, the unit normal vector;
- $\vec{t}$, the unit tangent vector.

These two vectors provide a local linear model of the circle and its surrounding space. Think about the two vectors by imagining that we start at X and walk h units in either direction:

- $X + h\vec{n}$ is a point h units away from the linearized circle;

- $X + h\vec{t}$ is a point h units along the linearized circle.

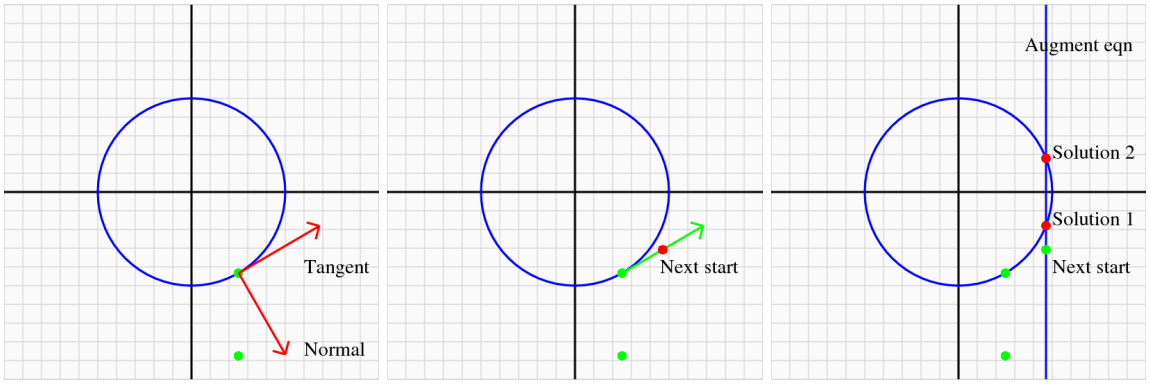
So, at least in the linear model, the normal vector takes us away from the curve, but the tangent vector tries to keep us on it.

How do we get access to these two vectors? Well, the normal vector is available by differentiating $f(x, y)$. For the circle, by differentiating $f(x, y)$ and normalizing, we have

$$\vec{n} = \begin{pmatrix} x \\ y \end{pmatrix}$$

The tangent vector is determined from its property that it must be orthogonal to the normal vector. In this case, we can see by inspection that the tangent vector is (up to a sign change):

$$\vec{t} = \begin{pmatrix} y \\ -x \end{pmatrix}$$



This suggests that, given the solution point X, we could search for the "next" solution point by using the starting value $X_0 = X + h\vec{t}$, and applying Newton's method to this new system. To apply Newton's method, we need a square system; the last equation is a somewhat arbitrary constraint that we supply. What equation should be supply for the new point? Our first point was found by the constraint $x = 0.5$, that is, we fixed the first coordinate. Can't we do that again? We have just computed a starting point:

$$X_0 = \begin{pmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{pmatrix} + h \begin{pmatrix} \frac{\sqrt{3}}{2} \\ -\frac{1}{2} \end{pmatrix}$$

so we can fix the first coordinate again, giving Newton the system:

$$\begin{aligned} x^2 + y^2 &= 1 \\ x &= \frac{1}{2} + h\frac{\sqrt{3}}{2} \end{aligned}$$

This is a square system, and if h is small enough, our starting point will be close to the circle, and the iteration should converge.

However, if we repeat this process several times, or with large values of h, we will eventually run into trouble. Although the tangent vector may be pointing in the positive x direction, the circle doesn't go past $x = 1$. So as our sequence of point samples approaches the turning point, we will find either that Newton's

method can't find a solution (there isn't one!) or that we have to decrease h so much that the solution we recover is extremely close to where we started, and hence uninteresting.

Interestingly enough, there is a warning about what is happening; we just haven't looked at it. What happens to the tangent vector as we approach $x = 1$? Obviously, the tangent begins to turn more and more vertical; in fact it becomes exactly vertical at the limit point and then reverses direction. The fact that the x component of the tangent is rapidly diminishing to zero, while the y component is increasing suggests that the curve is better parameterized by y than by x. In other words, at some point, our auxilliary constraint equation should be used to hold y fixed rather than x.

This suggests that, since the augmenting equation is completely arbitrary, we are free to choose to hold either x or y fixed, and to decide which to do, we should compare the corresponding components of the tangent vector t. If the linearized circle is almost horizontal, the x component will be large, but if it is becoming vertical, the y component will be the dominant one. Thus, the tangent vector can tell us whether to fix x or y when we create our augmenting equation for the Newton method. As long as we don't take enormous steps in h, such an approach will allow us to go around the circle as many times as we like.

More importantly, it suggests a general procedure for tracing out any connected curve in 2D that is defined by a differential implicit function, and we will see that it also indicates what we should do if we want to investigate problems in higher dimensions.

3 A Program for Continuation in 2D

We can sketch the form of a program to use continuation to find 2D solution points satisfying the implicit equation $f(x, y) = 0$. For convenience, we will want to define an *augmented* set of equations, $g(x, y) = 0$, defined by

$$g(x,y) = \begin{pmatrix} f(x,y) \\ X(p) - \alpha \end{pmatrix}$$

where the second equation allows us to hold either x or y fixed during a particular Newton iteration. For the circle problem, adding this equation is equivalent to drawing a vertical or horizontal line; if the line crosses the circle, then this defines the acceptable solutions that the Newton method will seek. It's already clear that picking a bad value of α makes the problem unsolvable. It may not be quite so clear that the choice of p should be made to select the variable which is most suitable as the fixed parameter; this is equivalent to saying that the p-th component of the tangent vector to the local solution curve is largest.

Corresponding to our augmented function, we have an augmented jacobian:

$$gp(x,y) = \begin{pmatrix} f_x(x,y) & f_y(x,y) \\ \delta_{1,p} & \delta_{2,p} \end{pmatrix}$$

Once we have defined the **g()** and **gp()** quantities, we can write a **newton()** function which is given a starting solution estimate and determines a solution to $g(x, y) = 0$, in other words, a point that is on the curve, and has the desired value of $X(p)$.

```
function [ status, x ] = newton ( n, x0, p, f, fp, tol )
  alpha = x0(p);
  x = x0;
  it = 0;
  it_max = 10;
  while ( 1 )
    if ( it_max < it )
      status = 1;
      return;
    end
    fx = f ( n, x );
```

```
    fx(n,1) = x(p) - alpha;
    fx_norm = max ( abs ( fx ) );
    if ( fx_norm <= tol )
      status = 0;
      return;
    end
    it = it + 1;
    fpx = fp ( n, x );
    fpx(n,1:n) = 0.0;
    fpx(n,p) = 1.0;
    dx = - fpx \ fx;
    x0 = x;
    x = x0 + dx;
  end
  return
end
```

Once we have a point x_0, we need to compute a "neighbor" point that is not too close. To do this, we use the unit tangent vector $\vec{t}$ and a stepsize h to compute

$$x_1 = x_0 + h \cdot \vec{t}$$

and then apply Newton's method to determine a point x_2 which satisfies the implicit function, and has the same value of the p-coordinate as x_1. To do all this, we need to compute the tangent vector. Any tangent vector t is a nonzero vector that is perpendicular to the normal vector. That is, such a vector satisfies:

$$f_x t_1 + f_y t_2 = 0$$

This is actually an $n-1$ by n linear system; once again, we need to augment our problem in order to use standard techniques. But if p is a reasonable choice for the parameter, it turns out that this means that the augmented Jacobian matrix is not singular, and we can set up a linear system of the form

$$\begin{pmatrix} f_x(x,y) & f_y(x,y) \\ \delta_{1,p} & \delta_{2,p} \end{pmatrix} \cdot \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Using a 1 on the right hand side just guarantees that whatever tangent vector we compute will not be the null vector. And once we compute t, we can normalize it. There is still an ambiguity as to the sign of the tangent vector, but we can determine that by requiring that it should have a nonnegative projection with respect to the *previous* tangent vector. If we can do all that, then we have a tangent vector of the size and direction we need in order to estimate the location of the next continuation point:

```
function [ t2, p2 ] = tangent ( n, x, t1, p, fp )
  fpx = fp ( n, x );
  fpx(n,1:n) = 0.0;
  fpx(n,p) = 1.0;
  b = zeros ( n, 1 );
  b(n) = 1.0;
  t2 = fpx \ b;
  t2_norm = norm ( t2 );
  t2 = t2 / t2_norm;
  [ ~, p2 ] = max ( abs ( t2 ) );
  if ( t2' * t1 < 0.0 )
    t2 = - t2;
  end
  return
end
```

To actually carry out a single step of continuation, we can write a function like the following:

```
function [ status, x2, t2, p2 ] = step ( n, x0, t0, p0, f, fp, h, tol )
```

```
  [ t2, p2 ] = tangent ( n, x0, t0, p0, fp );
  x1 = x0 + h * t2;
  [ status, x2 ] = newton ( n, x1, p0, f, fp, tol );
  return
end
```

The continuation process itself begins with a point that might not even be on the curve. It then will call **step()** over and over again. A sample calculation for the circle problem might look like this:

```
  step_max = 35;
  step_num = 0;
  xy = zeros ( 2, step_max );
  n = 2;
  x0 = [ 0.5; -2.0 ];
  p0 = 1;
  tol = 1.0E-05;

  fx_norm = max ( abs ( f_circle ( n, x0 ) ) );
  fprintf ( 1, ' %2d  %14.6g  %14.6g  %8.2e\n', 0, x0(1), x0(2), fx_norm );

  [ status, x2 ] = newton ( n, x0, p0, @f_circle, @fp_circle, tol );

  step_num = step_num + 1;
  fx_norm = max ( abs ( f_circle ( n, x2 ) ) );
  fprintf ( 1, ' %2d  %14.6g  %14.6g  %8.2e\n', step_num, x2(1), x2(2), fx_norm );

  x0 = x2;
  t0 = zeros ( n, 1 );
  h = 0.15;

  for step_num = 2 : step_max

    [ status, x2, t2, p2 ] = step ( n, x0, t0, p0, @f_circle, @fp_circle, h, tol );

    if ( status ~= 0 )
      fprintf ( 1, '\n' );
      fprintf ( 1, ' STEP failed!\n' );
      break
    end

    xy(1:2,step_num) = x2(1:2,1);
    fx_norm = max ( abs ( f_circle ( n, x2 ) ) );
    fprintf ( 1, ' %2d  %14.6g  %14.6g  %8.2e\n', step_num, x2(1), x2(2), fx_norm );

    x0 = x2;
    t0 = t2;
    p0 = p2;
  end
```

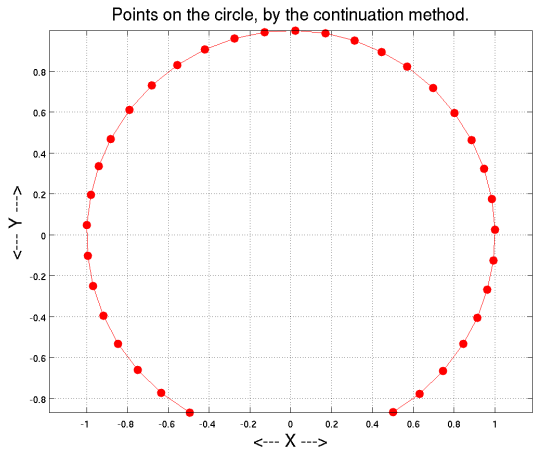
If we save the points in an array, or plot them as we go, we will get a nice outline of points that sample (part of) the circle, just as we hoped.

The Matlab codes discussed here are available online at:
https://people.sc.fsu.edu/~jburkardt/m_src/continuation/continuation.html

4 Where Can We Go From Here?

In this introductory discussion, we have kept things quite simple, but here are some issues to think about.

For the circle problem, note that the first row of the augmented jacobian is exactly zero at (0,0). Thus, if we tried to start the process with this (admittedly very bad) point, we couldn't even get started.



When we are computing the tangent vector, we use the same augmented jacobian as for the Newton method, but then we have to check that the tangent vector has the correct sign. We could, instead, use the previous tangent vector as the last row of the augmented matrix, and then solving the linear system would automatically guarantee that the two tangents had positive dot product.

Instead of a circle, suppose we had a sort of heart-shaped curve. Then the continuation algorithm would have great difficulties near the cusp in the curve, because the jacobian will go singular there too.

When we move to a problem in N dimensions, we still have N-1 equations in N unknowns. Assuming that the equations are "reasonable", this will define a 1-dimensional differentiable curve in N-dimensional space, so we can extend the methods we used in 2 dimensions to this new problem.

If the Newton iteration fails, we have many things we could try in order to keep going. We could vary the parameter choice P, or we decrease the continuation stepsize h.

The continuation method is occasionally used in order to solve a system of N "difficult" nonlinear equations $F(X) = 0$. This is attempted by creating a simple nonlinear equation $G(X) = 0$ whose solution is known, and then setting up the following problem in N+1 dimensional space:

$$H(X,t) = t * F(X) + (1-t) * G(X)$$

We then do "continuation" on the function $H(X,t)$, starting at $t = 0$, and seeking to compute the value of the curve when $t = 1$. This is sometimes called a *homotopy* solution. Many things can happen between $t = 0$ and $t = 1$ to make this approach fail, but it has been used to solve problems for which a direct assault on $G(X)$ fails.

Note that the continuation method has some interesting relations to the problem of solving a system of ordinary differential equations (ODE's). In particular, when we take a continuation step, we essentially choose one parameter to play the role of "time"; the tangent vector is then equivalent to the derivative function that appears on the right hand side of the ODE. But after we take a sort of Euler step, we actually have access to the implicit function, and so the subsequent Newton iteration can force our local error to go to zero, something impossible for ODE solvers.

References

[1] Cor den Heijer, Werner Rheinboldt, *On Steplength Algorithms for a Class of Continuation Methods*,

SIAM Journal on Numerical Analysis, Volume 18, Number 5, October 1981, pages 925-947.

[2] Milan Kubicek, *Algorithm 502: Dependence of solution of nonlinear systems on a parameter*, ACM Transactions on Mathematical Software, Volume 2, Number 1, March 1976, pages 98-107.

[3] Werner Rheinboldt, *Solution Field of Nonlinear Equations and Continuation Methods*, SIAM Journal on Numerical Analysis, Volume 17, Number 2, April 1980, pages 221-237.

[4] Werner Rheinboldt, *Numerical Analysis of Continuation Methods for Nonlinear Structural Problems*, Computers and Structures, Volume 13, 1981, pages 103-114.

[5] Werner Rheinboldt, John Burkardt, *A Locally Parameterized Continuation Process*, ACM Transactions on Mathematical Software, Volume 9, Number 2, June 1983, pages 215-235.

[6] Werner Rheinboldt, John Burkardt, *Algorithm 596: A Program for a Locally Parameterized Continuation Process,* ACM Transactions on Mathematical Software, Volume 9, Number 2, June 1983, pages 236-241.

[7] Werner Rheinboldt, **Numerical Analysis of Parameterized Nonlinear Equations**, Wiley, 1986, ISBN: 0-471-88814-1, LC: QA372.R54.

[8] Jan Verschelde, *Algorithm 795: PHCPACK: A general-purpose solver for polynomial systems by homotopy continuation*; ACM Transactions on Mathematical Software, Volume 25, Number 2, June 1999, page 251.

[9] Layne Watson, Dan Fenner *Algorithm 550: Chow-Yorke algorithm for fixed points or zeros of C2 maps*, ACM Transactions on Mathematical Software, Volume 6, Number 2, June 1980, pages 252-259.

[10] Layne Watson, Stephen Billups, Alexander Morgan, *Algorithm 652: HOMPACK: globally convergent homotopy algorithms, for finding zeros or fixed points of nonlinear systems of equations*, ACM Transactions on Mathematical Software, Volume 13, Number 3, September 1987, pages 281-310.

[11] Layne Watson, Masha Sosonkina, Robert Melville, Alexander Morgan, Homer Walker, *Algorithm 777: HOMPACK90: A Suite of Fortran 90 Codes for Globally Convergent Homotopy Algorithms*; ACM Transactions on Mathematical Software, Volume 23, Number 4, December 1997, page 514.

[12] Steven Wise, Andrew Sommese, Layne Watson, *Algorithm 801: POLSYS_PLP: A Partitioned Linear Product Homotopy Code for Solving Polynomial Systems of Equations*, ACM Transactions of Mathematical Software, Volume 26, Number 1, March 2000, page 176.

[13] Hai-Jun Su, Michael McCarthy, Masha Sosonkina, Layne Watson, *Algorithm 857: POLSYS_GLP: A Parallel General Linear Product Homotopy Code for Solving Polynomial Systems of Equations*, ACM Transactions on Mathematical Software, Volume 32, Number 4, December 2006, page 561.