Notes for 2017-04-21

Pacing the Path

So far, we have focused on nonlinear equations ($f : \mathbb{R}^n \to \mathbb{R}^n$), and to a lesser extent on optimization problems ($\phi : \mathbb{R}^n \to \mathbb{R}$). Often, though, nonlinear equations and optimization problems depend on some extra parameter. For example:

- In fitting problems, we care about the solution as a function of a regularization parameter.
- In population biology, we care about equilibrium population levels as a function of various parameters: birth rates, death rates, initial population sizes, etc.
- In mechanics, we care about the deformation of a structure as a function of load.
- In chemical kinetics, we care about equilibrium chemical concentrations as a function of temperature.
- In engineering problems involving a tradeoff between two parameters (e.g. mass and stiffness), we care about the optimal setting of one parameter given a fixed value of the other.
- In stochastic problems, we may care about the behavior as a function of the variance of the noise term, or perhaps as a function of an autocorrelation time.

For these types of problems, *continuation* strategies are often a good choice. The basic picture in a continuation strategy for solutions of an equation $f(x(s), s) = 0$ where $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ starting from some easily computed solution $x(s_0)$ is:

- Given $x(s_j)$, choose a new $s' = s_j + \Delta s$.
- *Predict* $x(s')$ based on the behavior at s. Two common predictors are
  - *Trivial*: Guess $x(s') \approx x(s_j)$.

  - *Euler:* Guess $x(s') \approx x(s_j) - \frac{\partial f}{\partial s}(x_j, s_j)^{-1} \frac{\partial f}{\partial s}(x_j, s_j)$.

- *Correct* by taking a few steps of Newton iteration.
- Either *accept* $s_{j+1} = s'$ and a corresponding $x(s_j)$ if the Newton iteration converged, or try again with a smaller Δs. If the Newton iteration converges very quickly, we may increase Δs.

Continuation is also natural if we really do care about a problem with no free parameters, but we lack a good initial guess with which to start an iterative method to solve the problem. In that case, a reasonable strategy is often to *introduce* a parameter s such that the equations at $s = 0$ are easy and the equations at $s = 1$ are the ones that we would like to solve. Such a constructed path in problem space is sometimes called a *homotopy*. In many cases, one can show that solutions are continuous (though not necessarily differentiable) functions of the homotopy parameter, so that following a homotopy path with sufficient care can provide *all* solutions even for hard nonlinear problems. For this reason, homotopy methods are particularly effective for solving systems of polynomial equations. Another very popular family of homotopy methods are the interior point methods for constrained optimization problems, which we will touch on briefly next week.

Tough to Trace

As a starting example, let's consider a variation on the equation from one of our first nonlinear systems lectures, a discretization of the thermal blowup equation

$$\frac{d^2u}{dx^2} + \exp(\gamma u) = 0$$

subject to $u(0) = u(1) = 0$. We approximate the derivative with a mesh to get a system of equations of the form

$$h^{-2} \left(u_{j-1} - 2u_j + u_{j+1} \right) + \exp(\gamma u_j) = 0$$

where u_j is the approximate solution at a mesh point $x_j = jh$ with $h = 1/(N+1)$. The boundary conditions are $u_0 = u_{N+1} = 0$, and the difference equations govern the behavior on the interior. Compared to the last time we saw this system, though, we have introduced a new feature: the rate constant γ.

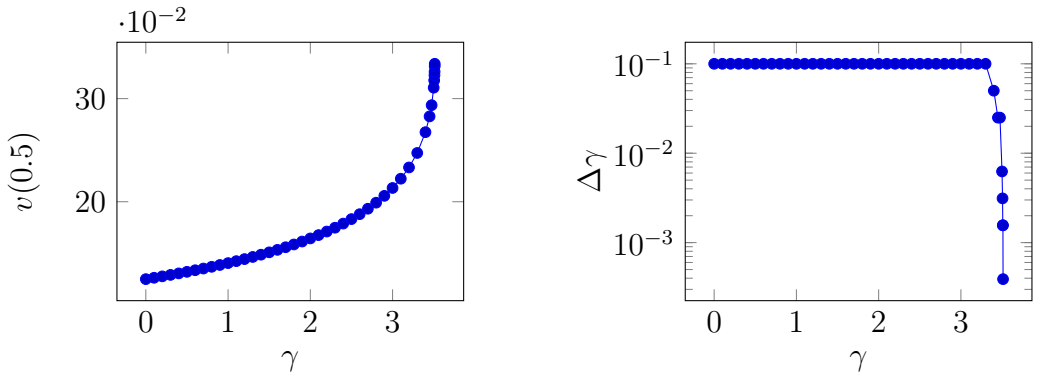

Figure 1: Center solution and step size versus γ

When γ is equal to zero, the differential equation becomes

$$\frac{d^2u}{dx^2} + 1 = 0,$$

which has the solution (subject to boundary conditions)

$$u(x;0) = \frac{1}{2}x(1-x).$$

For larger values of γ, things become more interesting. Based on physical reasoning, we expect the solutions to get more unstable (and harder) as γ grows. We therefore consider a strategy in which we incrementally increase γ, at each point using a trivial predictor (the solution for the previous γ) as an initial guess for a Newton iteration. If the Newton iteration does not converge in a few steps, we try again with a smaller step, stopping once the step size has become too small.

The behavior of the algorithm as a function of γ is shown in Figure 1. As we get just past $\gamma = 3.5$, the solution becomes more and more sensitive to small changes in γ, and we have to take shorter steps in order to get convergence. This is reflective of an interesting physical phenomenon known (a bifurcation). Mathematically, what we see is the effect of the Jacobian becoming closer and closer to singular at the solution.

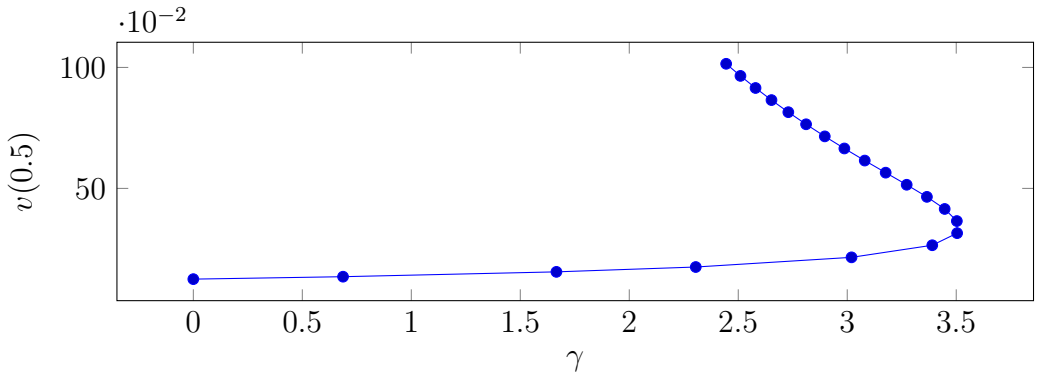

Figure 2: Center solution versus γ traced by continuation in the midpoint value.

Picking Parameters

In the previous section, we saw that continuation allowed us to march γ right up to some critical parameter, but not beyond. We can get a clearer picture of what is going on — and better solver stability — if we look at the same problem as a function of a *different parameter*. In particular, let us consider controlling the midpoint value μ, and letting both v and γ be implicit functions of the midpoint value. That is, we have the equations

$$F(v, \gamma; \mu) = \begin{bmatrix} -h^{-2} T_N v + \exp(\gamma v) \\ e_{\text{mid}}^T v - \mu \end{bmatrix} = 0$$

with the Jacobian matrix (with respect to v and γ)

$$\frac{\partial F}{\partial (v, \gamma)} = \begin{bmatrix} -h^2 T_n + \gamma \operatorname{diag}(\exp(\gamma v)) & v \odot \exp(\gamma v) \\ e_{\text{mid}}^T & 0 \end{bmatrix}$$

where we use $a \odot b$ to denote elementwise multiplication. If we use the same continuation process with a trivial predictor to trace out the behavior of the midpoint as a function of γ, we obtain Figure 2. This picture makes the behavior of the solution close to $\gamma = 3.5$ a little more clear. The phenomenon shown is called a *fold bifurcation*. Physically, we have that for $\gamma \lesssim 3.5$, there are two distinct solutions (one stable and one unstable); as γ increases, these two solutions approach each other until at some critical γ value they "meet." Beyond the critical value, there is no solution to the equations.

Pseudoarclength Ideas

What if we think we might run into a fold bifurcation, but do not know a good alternate parameter for continuation? A natural idea is to parameterize the solution curve (e.g. $(v(\gamma), \gamma)$) in terms of an *arclength* parameter. In practice, we do not care too much about exactly controlling arclength; it is just a mechanism to avoid picking parameters. Therefore, we pursue *pseudo-arclength* strategies as an alternative.

For the simplest pseudo-arclength continuation strategy, consider a function $F : \mathbb{R}^{n+1} \to \mathbb{R}^n$. Assuming the Jacobian has maximal rank, we expect there to be a solution curve $x : \mathbb{R} \to \mathbb{R}^n$ such that $F(x(s)) = 0$. The null vector of the Jacobian F' is tangent to x, and so we can use this to predict a new point. The basic procedure to get a new point on the curve starting from x^j is then:

- Consider the Jacobian $F'(x^j) \in \mathbb{R}^{n\times(n+1)}$ and compute a null vector v (a simple approach is to compute a QR factorization). Choose a tangent vector $t^j \propto v$; usually we normalize so that $t^{j-1} \cdot t^j > 0$.

- Move a short distance along the tangent direction (Euler predictor), or otherwise predict a new point.

- Correct back to the curve by the iteration

  $$y^{k+1} = y^k - F'(y^k)^\dagger F(y^k)$$

  where $F'(y^k)^\dagger \in \mathbb{R}^{(n+1)\times n}$ is the pseudoinverse of the Jacobian. This is equivalent to solving the problem

  $$\text{minimize } \|p^k\|^2 \text{ s.t. } F'(y^k)p^k = -F(y^k).$$

  The steps of this underdetermined system should quickly take us back to a new point on the curve.

- If the iteration curves and the new point is OK, accept the point and move on. Otherwise, reject the point and try again with a shorter step in the tangent direction.

And Points Beyond

There is a large and fascinating literature on numerical continuation methods and on the numerical analysis of implicitly defined functions. Beyond the predictor-corrector methods that we have described, there are various other methods that address similar problems: piecewise linear (simplex) continuation, pseudo-transient continuation, and so forth. We can combine continuation ideas with all the other ideas that we have described in the course; for example, one can do clever things with Broyden updates as one walks along the curve. We can also apply step control techniques that some of you may have learned in a class like CS 4210 in the context of methods for solving ordinary differential equations.

A little knowledge of continuation methods can take you a long way, but if you would like to know more, I recommend *Introduction to Numerical Continuation Methods* by Allgower and Georg.

Appendix: Codes

Continuation in γ

```
function [V,x,gammas] = blowupc

  % -- Number of (interior) mesh points and mesh spacing
  N = 50;            % Number of (interior) mesh points
  h = 1/(N+1);       % Mesh spacing
  x = linspace(0,1,N+2)'; % Mesh points

  % -- Set up some things for solver
  e = ones(N,1);

  % -- Initial value for gamma and solution at gamma = 0
  gamma = 0;
  v = x .* (1-x)/2;
  V = v;
  gammas = [gamma];

  % -- Continuation loop
  dgamma = 0.1;
  vprev = v;

  while (dgamma > 1e-4) & (gamma < 100)
```

```
    gamma_prev = gamma;
    gamma = gamma + dgamma;

    % -- Newton loop (fixed number of steps)
    for k = 1:5

      % Compute residual vector and record norm
      r = ( v(1:N)-2*v(2:N+1)+v(3:N+2) )/h^2 + exp(gamma * v(2:N+1));

      % Form (sparse) Jacobian
      J = spdiags([e/h^2, gamma*exp(gamma * v(2:N+1))-2/h^2, e/h^2], -1:1, N,N);

      % Compute Newton update
      p = J\r;
      v(2:N+1) = v(2:N+1)-p;

      % Quit if we are making small changes
      if norm(p) < 1e-10 * norm(v), break; end
    end

    % -- Accept step or reject and try a shorter step
    if norm(p) < 1e-10 * norm(v)
      V = [V, v];
      gammas = [gammas, gamma];
      vprev = v;
    else
      v = vprev;
      gamma = gamma_prev;
      dgamma = dgamma / 2;
    end

  end
```

Continuation in μ

```
function [V,x,gammas] = blowupmid

  % -- Number of (interior) mesh points and mesh spacing
  N = 50;            % Number of (interior) mesh points
  h = 1/(N+1);       % Mesh spacing
  x = linspace(0,1,N+2)'; % Mesh points

  % -- Set up some things for solver
  e = ones(N,1);

```

```
% -- Initial value for gamma and solution at gamma = 0
gamma = 0;
v = x .* (1-x)/2;
vmid = 0.125;
V = v;
gammas = [gamma];

% -- Continuation loop
dvmid = 0.01;
vprev = v;
gamma_prev = gamma;

while (dvmid > 1e-4) & (vmid < 1)
  vmid_prev = vmid;
  vmid = vmid + dvmid;

  % -- Newton loop (fixed number of steps)
  for k = 1:5

    % Compute residual vector
    r = [( v(1:N)-2*v(2:N+1)+v(3:N+2) )/h^2 + exp(gamma * v(2:N+1));
        v(N/2+1)-vmid];

    % Form (sparse) Jacobian
    J = spdiags([e/h^2, gamma*exp(gamma * v(2:N+1))-2/h^2, e/h^2], -1:1, N,N);
    b = v(2:N+1) .* exp(gamma * v(2:N+1));
    c = zeros(1,N); c(N/2) = 1;

    % Compute Newton update
    p = [J, b; c, 0]\r;
    v(2:N+1) = v(2:N+1)-p(1:end-1);
    gamma = gamma-p(end);

    % Quit if we are making small changes
    if norm(p) < 1e-10 * norm(v)
      dvmid = min(2*dvmid, 0.05);
      break;
    end
  end

  % -- Accept step or reject and try a shorter step
  if norm(p) < 1e-10 * norm(v)
    V = [V, v];
    gammas = [gammas, gamma];
    vprev = v;
```

```
56        gamma_prev = gamma;
57      else
58        v = vprev;
59        gamma = gamma_prev;
60        vmid = vmid_prev;
61        dvmid = dvmid / 2;
62      end
63
64    end
```