CS 4220: Numerical Analysis

Solvers on an autocatalytic example

David Bindel

2026-04-06

An autocatalytic example

We now consider a more serious example problem, a nonlinear system lsthat comes from a discretized PDE reaction-diffusion model describing (for example) the steady state heat distribution in a medium going an auto-catalytic reaction. The physics is that heat is generated in the medium due to a reaction, with more heat where the temperature is higher. The heat then diffuses through the medium, and the outside edges of the domain are kept at the ambient temperature. The PDE and boundary conditions are

$$\begin{aligned} \frac{d^2v}{dx^2} + \lambda \exp(v) = 0, \quad x \in (0,1) \\ v(0) = v(1) = 0. \end{aligned}$$

For the moment, we will set $\lambda = 1$.

We discretize the PDE for computer solution by introducing a mesh $x_i = ih$ for $i = 0, \ldots, N+1$ and $h = 1/(N+1)$; the solution is approximated by $v(x_i) \approx v_i$. We set $v_0 = v_{N+1} = 0$; when we refer to v without subscripts, we mean the vector of entries v_1 through v_N. This discretization leads to the nonlinear system

$$f_i(v) \equiv \frac{v_{i-1} - 2v_i + v_{i+1}}{h^2} + \exp(v_i) = 0.$$

This system has two solutions; physically, these correspond to stable and unstable steady-state solutions of the time-dependent version of the model. We show the two figures in Figure 1.

To solve for a Newton step, we need both f and the Jacobian of f with respect to the variables v_j. This is a tridiagonal matrix, which we can write as

$$J(v) = -h^{-2}T_N + \operatorname{diag}(\exp(v))$$

where $T_N \in \mathbb{R}^{N \times N}$ is the tridiagonal matrix with 2 down the main diagonal and -1 on the off-diagonals.

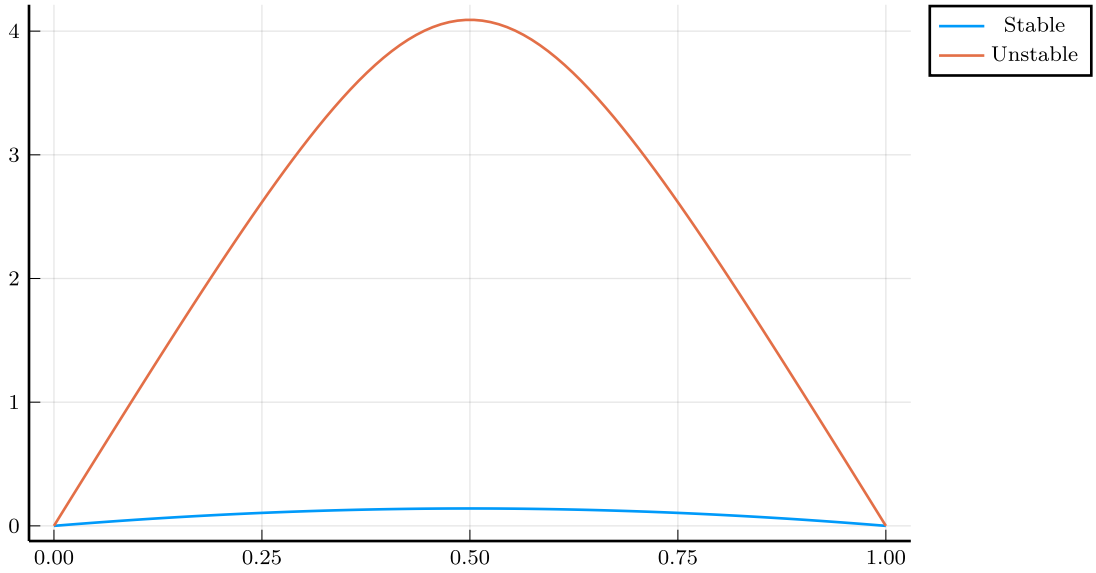

Figure 1: Stable and unstable solutions to autocatalytic equation.

```julia
function autocatalytic(v)
    N = length(v)
    fv = exp.(v)
    fv -= 2*(N+1)^2*v
    fv[1:N-1] += (N+1)^2*v[2:N  ]
    fv[2:N  ] += (N+1)^2*v[1:N-1]
    fv
end

function Jautocatalytic(v)
    N = length(v)
    SymTridiagonal(exp.(v) .- 2*(N+1)^2, (N+1)^2 * ones(N-1))
end
```

For an initial guess, we use $v_i = \alpha x_i(1 - x_i)$ for different values of α. For $\alpha = 0$, we converge to the stable solution; for $\alpha = 20$ and $\alpha = 40$, we converge to the unstable solution. We eventually see quadratic convergence in all cases, but for $\alpha = 40$ there is a longer period before convergence sets in. For $\alpha = 60$, the method does not converge at all.

For an initial guess, we use $v_i = \alpha x_i(1 - x_i)$ for different values of α. For $\alpha = 0$, we converge to the stable solution; for $\alpha = 20$ and $\alpha = 40$, we converge to the unstable solution (Figure 1). We eventually see quadratic convergence in all cases, but for $\alpha = 40$ there is a longer period before convergence sets in (Figure 2). For $\alpha = 60$, the method does not converge at all.

```julia
function newton_autocatalytic(α, N=100, nsteps=50, rtol=1e-8;
                              monitor=(v, resid)->nothing)
    v_all = [α*x*(1-x) for x in range(0.0, 1.0, length=N+2)]
    v = v_all[2:N+1]
    for step = 1:nsteps
        fv = autocatalytic(v)
        resid = norm(fv)
        monitor(v, resid)
        if resid < rtol
            v_all[2:N+1] = v
            return v_all
        end
        v -= Jautocatalytic(v)\fv
    end
    error("Newton did not converge after $nsteps steps")
end
```

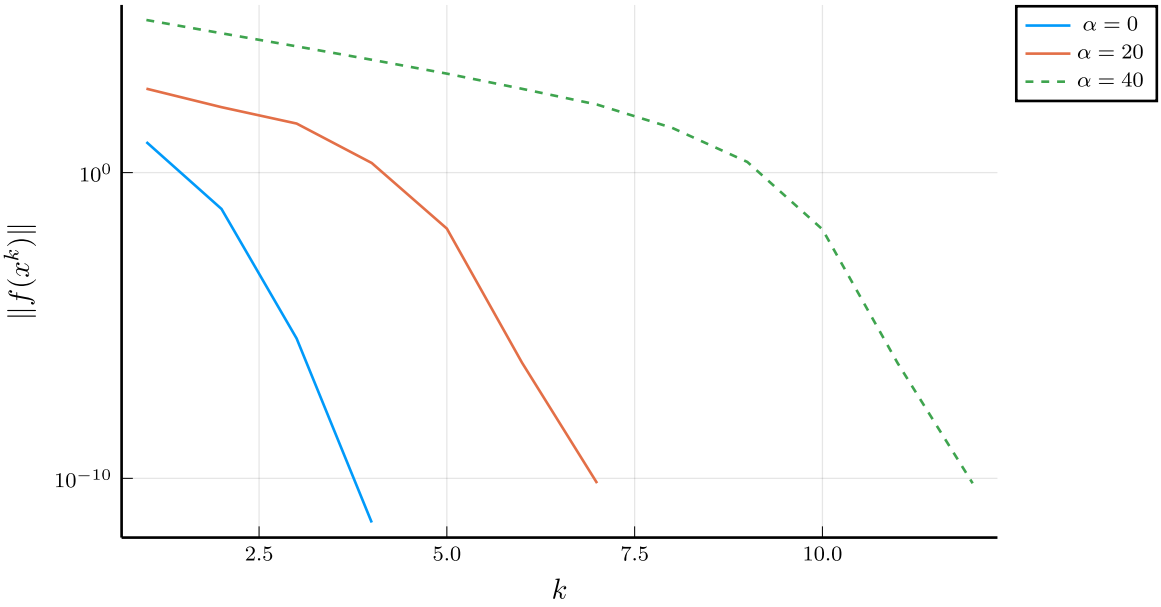

Figure 2: Convergence of Newton iteration for autocatalytic example

We can derive a Newton-like fixed point iteration from the observation that if v remains modest, the Jacobian is pretty close to $-h^2 T_N$. This gives us the iteration

$$h^{-2} T_N v^{k+1} = \exp(v^k).$$

In Figure 3, we compare the convergence of this fixed point iteration to Newton's method. The

fixed point iteration does converge, but it shows the usual linear convergence, while Newton's method converges quadratically.

```
function fp_autocatalytic(α, N=100, nsteps=500, rtol=1e-8;
                          monitor=(v, resid)->nothing)
    v_all = [α*x*(1-x) for x in range(0.0, 1.0, length=N+2)]
    v = v_all[2:N+1]
    TN = SymTridiagonal(2.0*ones(N), -ones(N-1))
    F = ldlt(TN)
    for step = 1:nsteps
        fv = autocatalytic(v)
        resid = norm(fv)
        monitor(v, resid)
        if resid < rtol
            v_all[2:N+1] = v
            return v_all
        end
        v[:] = F\(exp.(v)/(N+1)^2)
    end
    error("Fixed point iteration did not converge after $nsteps steps (α=$α)")
end
```

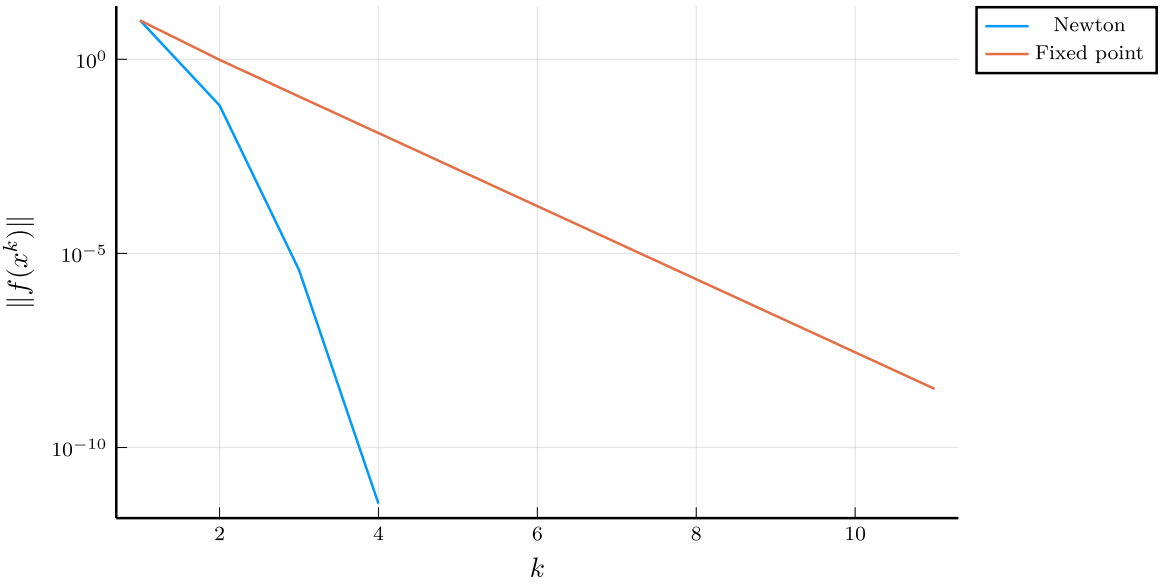

Figure 3: Convergence of Newton vs fixed point solvers.

Questions

1. Consider choosing $v = \alpha x(1-x)$ so that the equation is exactly satisfied at $x = 1/2$. How would you do this numerically?
2. Modify the Newton solver for the discretization of the equation $v'' + \lambda \exp(v) = 0$. What happens as λ grows greater than one? For what size λ can you get a solution? Try incrementally increasing λ, using the final solution for the previous value of λ as the initial guess at the next value.

Some practical issues

In general, there is no guarantee that a given solution of nonlinear equations will have a solution; and if there is a solution, there is no guarantee of uniqueness. This has a practical implication: many incautious computationalists have been embarrassed to find that they have "solved" a problem that was later proved to have no solution!

When we have reason to believe that a given system of equations has a solution — whether through mathematical argument or physical intuition — we still have the issue of finding a good enough initial estimate that Newton's method will converge. In coming lectures, we will discuss "globalization" methods that expand the set of initial guesses for which Newton's method converges; but globalization does not free us from the responsibility of trying for a good guess. Finding a good guess helps ensure that our methods will converge quickly, and to the "correct" solution (particularly when there are multiple possible solutions).

We saw one explicit example of the role of the initial guess in our analysis of the discretized blowup PDE problem. Another example occurs when we use unguarded Newton iteration for optimization. Given a poor choice of initial guess, we are as likely to converge to a saddle point or a local maximum as to a minimum! But we will address this pathology in our discussion of globalization methods.

If we have a nice problem and an adequate initial guess, Newton's iteration can converge quite quickly, at least in terms of the number of iterations. But at each step, we need to form and solve a linear system, and this involves the usual linear algebra issues. Depending on the problem, we may need to think about ill-conditioned (or even singular) Jacobian matrices close to a solution. And even for well-behaved problems where the Jacobian matrices are not ill-conditioned, we may worry about the cost of forming the Jacobian and solving systems with if if N is large. Fortunately, large systems of nonlinear equations often have structure that leads to structure in the Jacobian matrix (such as the tridiagonal structure in our autocatalytic example). It is important to identify and use such structure when it is present.

Once we have an iteration that we are happy with, we still have to think about when we will be satisfied with an approximate solution. A robust solver should check a few possible termination criteria:

- *Iteration count*: It makes sense to terminate (possibly with a diagnostic message) whenever an iteration starts to take more steps than one expects — or perhaps more steps than one can afford. If nothing else, this is necessary to deal with issues like poor initial guesses.
- *Residual check*: We often declare completion when $\|f(x^k)\|$ is sufficiently close to zero. What is "close to zero" depends on the scaling of the problem, so users of black box solvers are well advised to check that any default residual checks make sense for their problems.
- *Update check*: Once Newton starts converging, a good estimate for the error at step x^k is $x^{k+1} - x^k$. A natural test is then to make sure that $\|x^{k+1} - x^k\|/\|x^{k+1}\| < \tau$ for some tolerance τ. Of course, this is really an estimate of the relative error at step k, but we will report the (presumably better) answer x^{k+1} – like anyone else who can manage it, numerical analysts like to have their cake and eat it, too.

A common problem with many solvers is to make the termination criteria too lax, so that a bad solution is accepted; or too conservative, so that good solutions are never accepted.

One common mistake in coding Newton's method is to goof in computing the Jacobian matrix. This error is not only very common, but also very often overlooked. After all, a good approximation to the Jacobian often still produces a convergent iteration; and when iterations diverge, it is hard to distinguish between problems due to a bad Jacobian and problems due to a bad initial guess. However, there is a simple clue to watch for that can help alert the user to a bad Jacobian calculation. In most cases, Newton converges quadratically, while "almost Newton" iterations converge linearly. If you think you have coded Newton's method and a plot of the residuals shows linear behavior, look for issues with your Jacobian code!