**2023-03-29**

## 1 Fixed points and contraction mappings

As discussed in previous lectures, many iterations we consider have the form

$$x^{k+1} = G(x^k)$$

where $G : \mathbb{R}^n \to \mathbb{R}^n$. We call $G$ a *contraction* on $\Omega$ if it is Lipschitz with constant less than one, i.e.

$$\|G(x) - G(y)\| \leq \alpha \|x - y\|, \quad \alpha < 1.$$

A sufficient (but not necessary) condition for $G$ to be Lipschitz on $\Omega$ is if $G$ is differentiable and $\|G'(x)\| \leq \alpha$ for all $x \in \Omega$.

According to the *contraction mapping theorem* or *Banach fixed point theorem*, when $G$ is a contraction on a closed set $\Omega$ and $G(\Omega) \subset \Omega$, there is a unique fixed point $x^* \in \Omega$ (i.e. a point such that $x^* - G(x^*)$). If we can express the solution of a nonlinear equation as the fixed point of a contraction mapping, we get two immediate benefits.

First, we know that a solution exists and is unique (at least, it is unique within $\Omega$). This is a nontrivial advantage, as it is easy to write nonlinear equations that have no solutions, or have continuous families of solutions, without realizing that there is a problem.

Second, we have a numerical method – albeit a potentially slow one – for computing the fixed point. We take the fixed point iteration

$$x^{k+1} = G(x^k)$$

started from some $x^0 \in \Omega$, and we subtract the fixed point equation $x^* = G(x^*)$ to get an iteration for $e^k = x^k - x^*$:

$$e^{k+1} = G(x^* + e^k) - G(x^*)$$

Using contractivity, we get

$$\|e^{k+1}\| = \|G(x^* + e^k) - G(x^*)\| \leq \alpha \|e^k\|$$

which implies that $\|e^k\| \leq \alpha^k \|e^0\| \to 0$.

When error goes down by a factor of $\alpha > 0$ at each step, we say the iteration is *linearly convergent* (or *geometrically convergent*). The name reflects a semi-logarithmic plot of (log) error versus iteration count; if the errors lie on a straight line, we have linear convergence. Contractive fixed point iterations converge at least linarly, but may converge more quickly.

## 1.1 A toy example

Consider the function $G : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ given by

$$G(x) = \frac{1}{4} \begin{bmatrix} x_1 - \cos(x_2) \\ x_2 - \sin(x_1) \end{bmatrix}$$

This is a contraction mapping on all of $\mathbb{R}^2$ (why?).

Let's look at $\|x^k - G(x^k)\|$ as a function of $k$, starting from the initial guess $x^0 = 0$ (Figure 1).

```
function test_toy_contraction()

    G(x) = 0.25*[x[1]-cos(x[2]); x[2]-sin(x[1])]

    # Run the fixed point iteration for 100 steps
    resid_norms = []
    x = zeros(2)
    for k = 1:100
        x = G(x)
        push!(resid_norms, norm(x-G(x), Inf))
    end

    x, resid_norms
end
```

### 1.1.1 Questions

1. Show that for the example above $\|G'\|_\infty \leq \frac{1}{2}$ over all of $\mathbb{R}^2$. This implies that $\|G(x) - G(y)\|_\infty \leq \frac{1}{2}\|x - y\|_\infty$.
2. The mapping $x \mapsto x/2$ is a contraction on $(0, \infty)$, but does not have a fixed point on that interval. Why does this not contradict the contraction mapping theorem?
3. For $S > 0$, show that the mapping $g(x) = \frac{1}{2}(x + S/x)$ is a contraction on the interval $[\sqrt{S}, \infty)$. What is the fixed point? What is the Lipschitz constant?

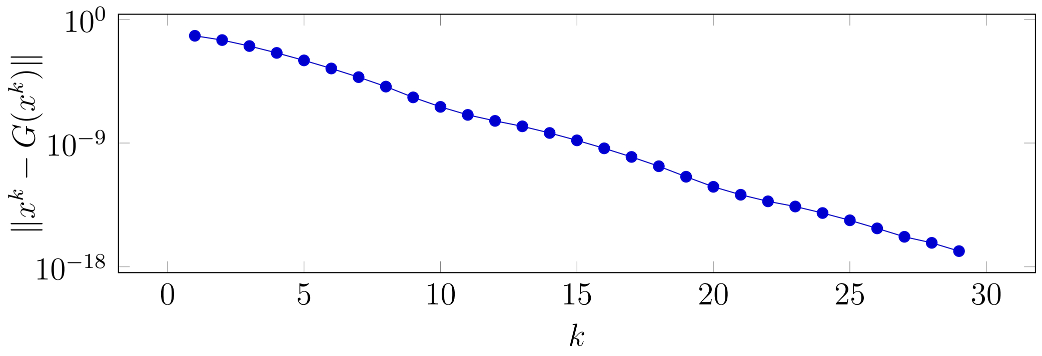

Figure 1: Convergence for `test_toy_contraction`.

## 2 Newton's method for nonlinear equations

The idea behind Newton's method is to approximate a nonlinear $f \in C^1$ by linearizations around successive guesses. We then get the next guess by finding where the linearized approximaton is zero. That is, we set

$$f(x^{k+1}) \approx f(x^k) + f'(x^k)(x^{k+1} - x^k) = 0,$$

which we can rearrange to

$$x^{k+1} = x^k - f'(x^k)^{-1} f(x^k).$$

Of course, we do not actually want to form an inverse, and to set the stage for later variations on the method, we also write the iteration as

$$\begin{aligned} f'(x^k)p^k &= -f(x^k) \\ x^{k+1} &= x^k + p^k. \end{aligned}$$

### 2.1 A toy example

Consider the problem of finding the solutions to the system

$$\begin{aligned} x + 2y &= 2 \\ x^2 + 4y^2 &= 4. \end{aligned}$$

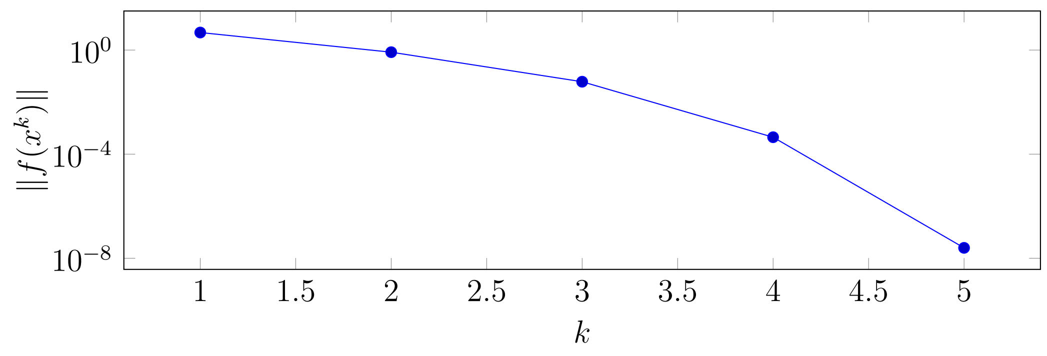

Figure 2: Convergence for `test_toy_newton` to $(0, 1)$.

That is, we are looking for the intersection of a straight line and an ellipse. This is a simple enough problem that we can compute the solution in closed form; there are intersections at $(0, 1)$ and at $(2, 0)$. Suppose we did not know this, and instead wanted to solve the system by Newton's iteration. To do this, we need to write the problem as finding the zero of some function

$$f(x, y) = \begin{bmatrix} x + 2y - 2 \\ x^2 + 4y^2 - 4 \end{bmatrix} = 0.$$

We also need the Jacobian $J = f'$:

$$\frac{\partial f}{\partial(x, y)} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix}.$$

We show the convergence of the Newton iteration in Figure 2.

```
function test_toy_newton(x, y)

    # Set up the function and the Jacobian
    f(x) = [x[1] + 2*x[2]-2;
            x[1]^2 + 4*x[2]^2 - 4]
    J(x) = [1      2     ;
            2*x[1] 8*x[2]]

    # Run ten steps of Newton from the initial guess
    x = [x; y]
    fx = f(x)
    resids = zeros(10)
    for k = 1:10
        x -= J(x)\fx
        fx = f(x)
        resids[k] = norm(fx)
    end

    x, resids
end
```

#### 2.1.1 Questions

1. Finding an (real) eigenvalue of $A$ can be posed as a nonlinear equation solving problem: we want to find $x$ and $\lambda$ such that $Ax = \lambda x$ and $x^T x = 1$. Write a Newton iteration for this problem.

## 2.2 Superlinear convergence

Suppose $f(x^*) = 0$. Taylor expansion about $x^k$ gives

$$0 = f(x^*) = f(x^k) + f'(x^k)(x^* - x^k) + r(x^k)$$

where the remainder term $r(x^k)$ is $o(\|x^k - x^*\|) = o(\|e^k\|)$. Hence

$$x^{k+1} = x^* + f'(x^k)^{-1} r(x^k)$$

and subtracting $x^*$ from both sides gives

$$e^{k+1} = f'(x^k)^{-1} r(x^k) = f'(x^k)^{-1} o(\|e^k\|)$$

If $\|f'(x)^{-1}\|$ is bounded for $x$ near $x^*$ and $x^0$ is close enough, this is sufficient to guarantee *superlinear convergence*. When we have a stronger condition, such as $f'$ Lipschitz, we get *quadratic convergence*, i.e. $e^{k+1} = O(\|e^k\|^2)$. Of course, this is all local theory – we need a good initial guess!

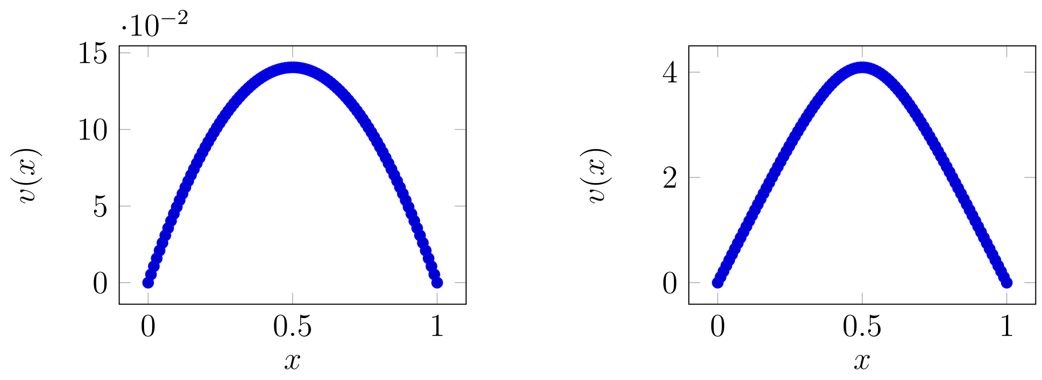

Figure 3: Two solutions for the blow-up example.

## 3 A more complex example

We now consider a more serious example problem, a nonlinear system that 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

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

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 solutions in Figure 3.

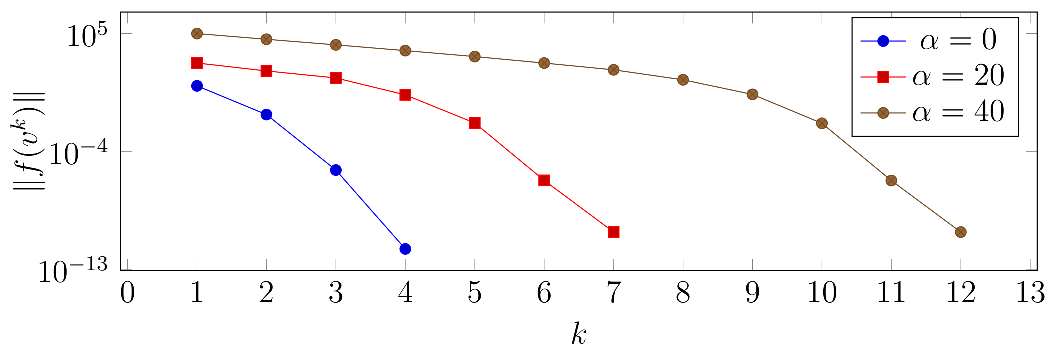

Figure 4: Newton convergence for different starting guesses.

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.

```
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 $\alpha$. For $\alpha = 0$, we converge to the stable solution; for $\alpha = 20$ and $\alpha = 40$, we converge to the unstable solution (Figure 3). We eventually see quadratic convergence in all cases, but for $\alpha = 40$ there is a longer period before convergence sets in (Figure 4). For $\alpha = 60$, the method does not converge at all.

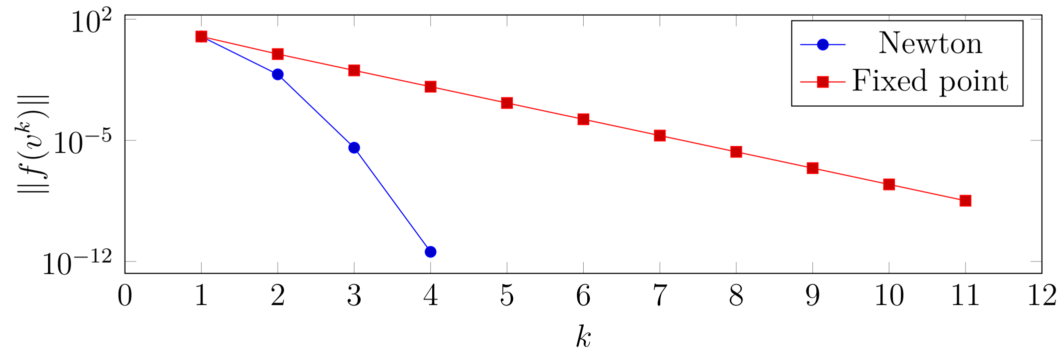

Figure 5: Newton convergence vs fixed point convergence.

```
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
```

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 5, 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
```

### 3.0.1 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 $\lambda$ grows greater than one? For what size $\lambda$ can you get a solution? Try incrementally increasing $\lambda$, using the final solution for the previous value of $\lambda$ as the initial guess at the next value.

## 4 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. But even then, 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 $\tau$. 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!