Bonus Lecture: Solving Systems of Equations

Chris Conlon

September 8, 2020

Grad IO

Basic Setup

Often we are interested in solving a problem like this:

Root Finding $f(x) = 0$

Optimization $\arg\min_x f(x)$.

These problems are related because we find the minimum by setting: $f'(x) = 0$

Linear Systems

Linear Systems

Many problems of the form: $A\mathbf{x} = b$.

- In general these are much easy to solve: $A^{-1}b = \mathbf{x}$.
- In practice the computer doesn't do this:
  1. Factorize into triangular matrices: QR, LU, or Cholesky (Positive Definite).
  2. Solve the transformed system of equations.
- This is what `backslash` in MATLAB does `A\b`.
- Or `np.linalg.solve(A,b)`.
- Takeaway: For stability and speed these are almost always preferred to $A^{-1}b$.
- If you need to solve many times for different b, saving the decomposition rather than A^{-1} is still usually a better idea.

Linear Systems: Iterative Methods

If A is really big you may not be able to invert it directly. Can still do it iteratively (less memory). Decompose matrix into three parts

- $A = D + L + U$ where D is diagonal and L, U are lower/upper triangles.
- Gauss Jacobi Iteration:

$$\mathbf{x}^{(n+1)} = D^{-1}\left(\mathbf{b} - (L+U)\mathbf{x}^{(n)}\right)$$

- Gauss-Seidel Iteration

$$\mathbf{x}^{(n+1)} = (L+D)^{-1}\left(\mathbf{b} - U\mathbf{x}^{(n)}\right)$$

- Richardson Iteration (with $\lambda < 1$):

$$x^{(n+1)} = x^{(n)} + \lambda\left(b - Ax^{(n)}\right)$$

Root Finding

Newton's Method for Root Finding

Consider the Taylor series for $f(x)$ approximated around $f(x_0)$:

$$f(x) \approx f(x_0) + f'(x_0) \cdot (x - x_0) + f''(x_0) \cdot (x - x_0)^2 + o_p(3)$$

Suppose we wanted to find a root of the equation where $f(x^*) = 0$ and solve for x:

$$\begin{aligned} 0 &= f(x_0) + f'(x_0) \cdot (x - x_0) \\ x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} \end{aligned}$$

This gives us an iterative scheme to find x^*:

1. Start with some x_n. Calculate $f(x_n), f'(x_n)$
2. Update using $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
3. Stop when $|x_{n+1} - x_n| < \epsilon_{tol}$.

Halley's Method for Root Finding

Consider the Taylor series for $f(x)$ approximated around $f(x_0)$:

$$f(x) \approx f(x_0) + f'(x_0) \cdot (x - x_0) + f''(x_0) \cdot (x - x_0)^2 + o_p(3)$$

Now let's consider the second-order approximation:

$$\begin{aligned} x_{n+1} &= x_n - \frac{2 f\left(x_n\right) f'\left(x_n\right)}{2\left[f'\left(x_n\right)\right]^2 - f\left(x_n\right) f''\left(x_n\right)} = x_n - \frac{f\left(x_n\right)}{f'\left(x_n\right) - \frac{f(x_n)}{f'(x_n)} \frac{f''(x_n)}{2}} \\ &= x_n - \frac{f\left(x_n\right)}{f'\left(x_n\right)}\left[1 - \frac{f\left(x_n\right)}{f'\left(x_n\right)} \cdot \frac{f''\left(x_n\right)}{2 f'\left(x_n\right)}\right]^{-1} \end{aligned}$$

- Last equation is useful because we only need to know $f(x_n)/f'(x_n)$ and $f''(x_n)/f'(x_n)$
- If we are lucky $f''(x_n)/f'(x_n)$ is easy to compute or ≈ 0 (Newton's method).

Root Finding: Convergence

How many iterations do we need? This is a tough question to answer.

- However we can consider convergence where $f(a) = 0$:

$$|x_{n+1} - a| \leq K_d * |x_n - a|^d$$

  - $d = 2$ (Newton's Method) quadratic convergence (we need $f'(x)$)
  - $d = 3$ (Halley's Method) cubic convergence (but we need $f''(x)$)

- Many implementations will benefit from damping at some rate $\lambda < 1$. $x_1 = x_0 - \lambda \frac{f(x_0)}{f'(x_0)}$ so that $\lambda = 1$ is a full Newton step.

Root Finding: Fixed Points

Some (not all) equations can be written as $f(x) = x$ or $g(x) = 0 : f(x) - x = 0$.

- In this case we can iterate on the fixed point directly

$$x_{n+1} = f(x_n)$$

- Advantage: we only need to calculate $f(x)$.
- There need not be a unique solution to $f(x) = x$.
- But... this may or may not actually work.

Contraction Mapping Theorem/ Banach Fixed Point

Consider a set $D \subset \mathbb{R}^n$ and a function $f : D \to \mathbb{R}^n$. Assume

1. D is closed (i.e., it contains all limit points of sequences in D)
2. $x \in D \implies f(x) \in D$
3. The mapping g is a contraction on D : There exists $q < 1$ such that

$$\forall x, y \in D : \quad \|f(x) - f(y)\| \leq q\|x - y\|$$

Then

1. There exists a unique $x^* \in D$ with $f(x^*) = x^*$
2. For any $x^{(0)} \in D$ the fixed point iterates given by $x^{(k+1)} := f\left(x^{(n)}\right)$ converge to x^* as $n \to \infty$
3. $x^{(n)}$ satisfies the a-priori error estimate $\left\|x^{(n)} - x^*\right\| \leq \frac{q^n}{1-q}\left\|x^{(1)} - x^{(0)}\right\|$
4. $x^{(n)}$ satisfies the a-posteriori error estimate $\left\|x^{(n)} - x^*\right\| \leq \frac{q}{1-q}\left\|x^{(n)} - x^{(n-1)}\right\|$

Some notes

- Not every fixed point relationship is a contraction.
- Iterating on $x_{n+1} = f(x_n)$ will not always lead to $f(x) = x$ or $g(x) = 0$.
- Convergence rate of fixed point iteration is slow or $q-$linear.
- When q is small this will be faster.
- q is sometimes called modulus of contraction mapping.
- A key example of a contraction: value function iteration!

Accelerated Fixed Points: Secant Method

Start with Newton's method and use the finite difference approximation

$$f'\left(x_{n-1}\right) \approx \frac{f\left(x_{n-1}\right) - f\left(x_{n-2}\right)}{x_{n-1} - x_{n-2}}$$

$$x_n = x_{n-1} - f\left(x_{n-1}\right) \frac{x_{n-1} - x_{n-2}}{f\left(x_{n-1}\right) - f\left(x_{n-2}\right)}$$

- This doesn't have the actual $f'(x_n)$ so it isn't quadratically convergent
- Instead is is superlinear with rate $q = \frac{1+\sqrt{5}}{2} = 1.618 < 2$ (Golden Ratio)
- Faster than fixed-point iteration but doesn't require computing $f'(x_n)$.
- Idea: can use past iterations to approximate derivatives and accelerate fixed points.
- For (inverse) quadratic approx: Brent's Method (sort of).

Accelerated Fixed Points: Anderson (1965) Mixing

Define the residual $r(x_n) = f(x_n) - x_n$. Find weights on previous k residuals:

$$\widehat{\alpha^n} = \arg\min_{\alpha} \left\| \sum_{k=0}^{m} \alpha_k^n \cdot r_{n-k} \right\| \text{ subject to } \sum_{k=0}^{m} \alpha_k^n = 1$$

$$x_{n+1} = (1-\lambda) \sum_{k=0}^{m} \widehat{\alpha_k^n} \cdot x_{n-k} + \lambda \sum_{k=0}^{m} \widehat{\alpha_k^n} \cdot f\left(x_{n-k}\right)$$

- Convex combination of weighted average of: lagged x_{n-k} and lagged $f(x_{n-k})$.
- Variants on this are known as Anderson Mixing or Anderson Acceleration.

Accelerated Fixed Points: SQUAREM (Varadhan and Roland 2008)

Define the residual $r(x_n) = f(x_n) - x_n$ and $v(x_n) = f \circ r\,(x_n) = f \circ f\,(x_n) - f\,(x_n)$.

$$\begin{aligned} x_{n+1} =& x_n & -2s\left[f\,(x_n) - x_n\right] & & +s^2\left[f \circ f\,(x_n) - 2f\,(x_n) + x_n\right] \\ =& x_n & -2sr & & +s^2(v - r) \end{aligned}$$

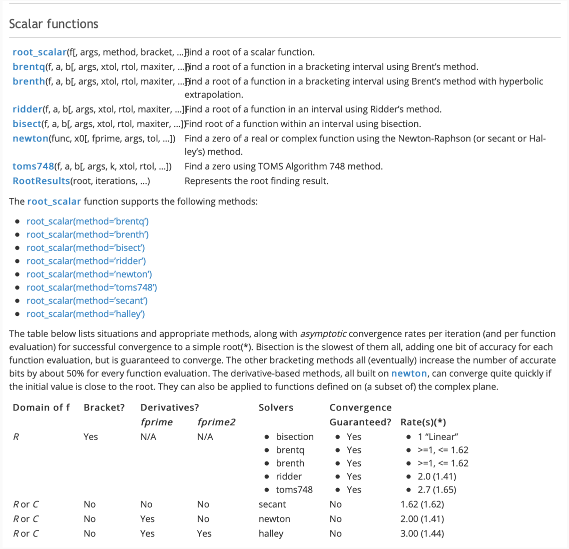
Three versions of stepsize:

$$s_1 = \frac{r^t r}{r^t(v-r)}, \quad s_2 = \frac{r^t(v-r)}{(v-r)^t(v-r)}, \quad s_3 = -\sqrt{\frac{r^t r}{(v-r)^t(v-r)}}$$

Idea: use two iterations to construct something more like the quadratic/Halley method.

Note: I am hand-waving, don't try to derive this.

NumPy



Scalar functions

root_scalar(f[, args, method, bracket, …])	Find a root of a scalar function.
brentq(f, a, b[, args, xtol, rtol, maxiter, …])	Find a root of a function in a bracketing interval using Brent's method.
brenth(f, a, b[, args, xtol, rtol, maxiter, …])	Find a root of a function in a bracketing interval using Brent's method with hyperbolic extrapolation.
ridder(f, a, b[, args, xtol, rtol, maxiter, …])	Find a root of a function in an interval using Ridder's method.
bisect(f, a, b[, args, xtol, rtol, maxiter, …])	Find root of a function within an interval using bisection.
newton(func, x0[, fprime, args, tol, …])	Find a zero of a real or complex function using the Newton-Raphson (or secant or Halley's) method.
toms748(f, a, b[, args, k, xtol, rtol, …])	Find a zero using TOMS Algorithm 748 method.
RootResults(root, iterations, …)	Represents the root finding result.

The **root_scalar** function supports the following methods:

- root_scalar(method='brentq')
- root_scalar(method='brenth')
- root_scalar(method='bisect')
- root_scalar(method='ridder')
- root_scalar(method='newton')
- root_scalar(method='toms748')
- root_scalar(method='secant')
- root_scalar(method='halley')

The table below lists situations and appropriate methods, along with *asymptotic* convergence rates per iteration (and per function evaluation) for successful convergence to a simple root(*). Bisection is the slowest of them all, adding one bit of accuracy for each function evaluation, but is guaranteed to converge. The other bracketing methods all (eventually) increase the number of accurate bits by about 50% for every function evaluation. The derivative-based methods, all built on **newton**, can converge quite quickly if the initial value is close to the root. They can also be applied to functions defined on (a subset of) the complex plane.

Domain of f	**Bracket?**	**Derivatives?**		**Solvers**	**Convergence**	
		fprime	***fprime2***		**Guaranteed?**	**Rate(s)(*)**
R	Yes	N/A	N/A	• bisection	• Yes	• 1 "Linear"
				• brentq	• Yes	• >=1, <= 1.62
				• brenth	• Yes	• >=1, <= 1.62
				• ridder	• Yes	• 2.0 (1.41)
				• toms748	• Yes	• 2.7 (1.65)
R or *C*	No	No	No	secant	No	1.62 (1.62)
R or *C*	No	Yes	No	newton	No	2.00 (1.41)
R or *C*	No	Yes	Yes	halley	No	3.00 (1.44)

In higher dimensions...

Multiple Equations

Solving $f(x) = 0$ for scalars is fine, but we often are interested in $F(\mathbf{x}) = 0$ or m nonlinear equations and k unknowns $\mathbf{x} = (x_1, \ldots, x_K) \in \mathbb{R}^k$.

- If we have $k > m$ we say the system is undetermined
- If we have $k < m$ we say the system is overdetermined
- I am going to focus on the square case of m equations and m unknowns.
- Think about a system of FOC for prices/quantities/etc.

For the most part, the approaches for scalar root finding still apply.

Solution Methods

General problem $F(\mathbf{x}) = 0$ or m nonlinear equations and m unknowns $\mathbf{x} = (x_1, \ldots, x_m) \in \mathbb{R}^m$.

$$\begin{array}{rcl} F_1(x_1, \ldots, x_m) & = & 0 \\ F_2(x_1, \ldots, x_m) & = & 0 \\ & \vdots & \\ F_{N-1}(x_1, \ldots, x_m) & = & 0 \\ F_N(x_1, \ldots, x_m) & = & 0 \end{array}$$

Solution Methods

Helpful to write $F(\mathbf{x}) = 0 \Leftrightarrow \mathbf{x} - \alpha F(\mathbf{x}) = \mathbf{x}$ which yields the fixed point problem:

$$G(\mathbf{x}) = \mathbf{x} - \alpha F(\mathbf{x})$$

Fixed point iteration

$$\mathbf{x^{n+1}} = G(\mathbf{x^n})$$

Nonlinear Richardson iteration or Picard iteration.

We need G to be a contraction mapping for iterative methods to guarantee a unique solution (often need strong monotonicity as well).

Gauss Jacobi: Simultaneous Best Reply

Current iterate: $\mathbf{x^n} = (x_1^n, x_2^n, \ldots, x_{m-1}^n, x_m^n)$.

Compute the next iterate x^{n+1} by solving one equation in one variable using only values from $\mathbf{x^n}$:

$$\begin{array}{rcl} F_1(x_1^{n+1}, x_2^n \ldots, x_{m-1}^n, x_m^n) & = & 0 \\ F_2(x_1^n, x_2^{n+1}, \ldots, x_{m-1}^n, x_m^n) & = & 0 \\ & \vdots & \\ F_{m-1}(x_1^n, x_2^n, \ldots, x_{m-1}^{n+1}, x_m^n) & = & 0 \\ F_m(x_1^n, x_2^n, \ldots, x_{m-1}^n, x_m^{n+1}) & = & 0 \end{array}$$

Requires contraction and strong monotonicity.

Gauss Seidel: Iterated Best Response

Current iterate: $\mathbf{x^n} = (x_1^n, x_2^n, \dots, x_{m-1}^n, x_m^n)$.

Compute the next iterate $\mathbf{x^{n+1}}$ by solving one equation in one variable updating as we go through:

$$\begin{aligned} F_1(x_1^{n+1}, x_2^n \dots, x_{m-1}^n, x_m^n) &= 0 \\ F_2(x_1^{n+1}, x_2^{n+1}, \dots, x_{m-1}^n, x_m^n) &= 0 \\ &\vdots \\ F_{m-1}(x_1^{n+1}, x_2^{n+1}, \dots, x_{m-1}^{n+1}, x_m^n) &= 0 \\ F_m(x_1^{n+1}, x_2^{n+1}, \dots, x_{m-1}^{n+1}, x_m^{n+1}) &= 0 \end{aligned}$$

Requires contraction and strong monotonicity.
You can speed things up (sometimes) by re-ordering equations.

Newton-Raphson Method

1. Take an initial guess $\mathbf{x^0}$
2. Take a Newton step by solving the following system of linear equations

$$J_F(\mathbf{x^n})\mathbf{s^n} = -F(\mathbf{x^n})$$

3. New guess $\mathbf{x^{n+1}} = \mathbf{x^n} + \mathbf{s^n}$ or $\mathbf{x^{n+1}} = \mathbf{x^n} - J_F^{-1}(\mathbf{x^n}) \cdot F(\mathbf{x^n})$
4. Good (Quadratic) Local convergence

- Requires J_F (Jacobian) to be Lipschitz continuous.
- Linearity means we do not need to take the inverse to solve the system (just QR decomp – `backslash` in MATLAB).
- Non-singularity of J_F is weaker than strong monotonicity (more like PSD).

Why not always do Newton-Raphson?

- Often computing or inverting $J_f(\mathbf{x^n})$ is hard.
- Alternatives focus on simplified ways to compute $J_f(\mathbf{x^n})$ or to update $J_f^{-1}(\mathbf{x^n})$
    - Some techniques similar to secant method (Broyden's Method).
    - Also what are known as quasi-Newton methods.
- If NR is feasible: start with that!

Broyden's Method

Idea: approximate the Jacobian $J_f(\mathbf{x^n}) \approx A_n$

1. Start with $A_0 = \mathbf{I}_m$.
2. Iterate on $\mathbf{x^{n+1}} = \mathbf{x^n} - A_n^{-1} F\left(\mathbf{x^n}\right)$
3. Update the Jacobian:

$$A_{n+1} = A_n - \frac{F\left(\mathbf{x^{n+1}}\right)\left[A_n^{-1} F\left(\mathbf{x^n}\right)\right]'}{\left[A_n F\left(\mathbf{x^n}\right)\right]'\left[A_n F\left(\mathbf{x^n}\right)\right]}$$

This is meant to be the multivariate version of the secant method.

Broyden-Fletcher-Goldfarb-Shanno (BFGS)

Same idea, different Jacobian update:

$$\begin{aligned} d_n &= \mathbf{x^{n+1}} - \mathbf{x^n} \\ g_n &= F(\mathbf{x^{n+1}}) - F(\mathbf{x^n}) \\ A_{n+1} &= A_n + \frac{g_n g_n'}{d_n' g_n} - \frac{A_n d_n d_n' A_n}{d_n' A_n d_n} \end{aligned}$$

Or the Davidson-Fletcher-Powell (DFP) version (operates directly on inverse)

$$A_{n+1}^{-1} = A_n^{-1} + \frac{d_n d_n'}{d_n' g_n} - \frac{A_n^{-1} g_n g_n' A_n^{-1}}{g_n' A_n^{-1} g_n}$$

Usually BFGS is preferred if you can invert A. Both of these preserve positive definiteness.

Derivative Free Methods

Most methods either calculate the derivative explicitly, or calculate it in course via multiple iterations. There are some exceptions:

- Powell's method.
- Nelder-Mead/Simplex.

There are some pathological problems for derivative/Jacobian based methods, but mostly these are hard to recommend.

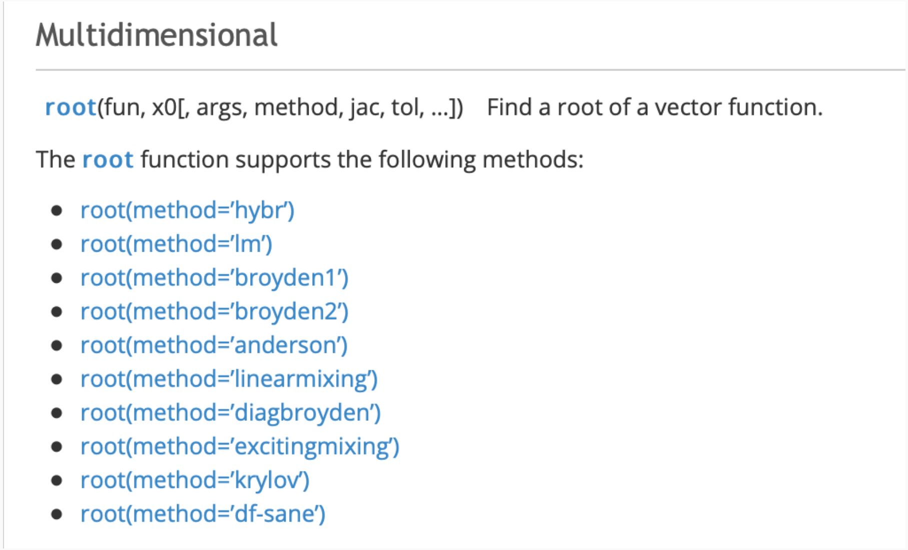
Least Squares Methods

Instead of solving $F(\mathbf{x}) = 0$, we could recast the problem as a least squares minimization problem:

$$\min_{\mathbf{x}} \sum_{m=1}^{M} (F_m(\mathbf{x}))^2$$

- Ex: Levenberg-Marquardt
- This works surprisingly well (including for overdetermined systems).
- It takes a convex combination of gradient steps and Newton steps.
- We will discus more of this when we talk about nonlinear optimization.

NumPy



Multidimensional

root(fun, x0[, args, method, jac, tol, ...])   Find a root of a vector function.

The root function supports the following methods:

- root(method='hybr')
- root(method='lm')
- root(method='broyden1')
- root(method='broyden2')
- root(method='anderson')
- root(method='linearmixing')
- root(method='diagbroyden')
- root(method='excitingmixing')
- root(method='krylov')
- root(method='df-sane')

The end