MA3227 Numerical Analysis II

Lecture 13: Simultaneous Nonlinear Equations

Simon Etter

2019/2020

Simultaneous Nonlinear Equations

Introduction

We have seen that bisection essentially solves the problem of scalar-valued nonlinear equations $f(x) = 0$. Complications only arise if we try to beat the already reasonable performance of bisection.

In this lecture, we will consider the problem of solving simultaneous nonlinear equations:

$$\text{Given a continuous function } f : \mathbb{R}^N \to \mathbb{R}^N \text{ with } N > 1,$$
$$\text{find } x \in \mathbb{R}^N \text{ such that } f(x) = 0.$$

Bad news: there is no analogue of bracketing intervals for $N > 1$. We thus no longer have bisection as a fall-back method, and also false position cannot be generalised to $N > 1$.

This is related to the fact that it is much harder to guarantee the existence of roots for $N > 1$ equations. In particular, there are no computationally verifiable certificates like bracketing intervals.

The following slides discuss how to generalise the Newton and secant methods to multiple nonlinear equations.

Simultaneous Nonlinear Equations

Newton's method

Given an initial guess x_0, construct the sequence

$$x_{k+1} = x_k - \nabla f(x_k)^{-1} f(x_k)$$

where $\nabla f(x)$ denotes the Jacobian matrix

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \dots & \frac{\partial f_1}{\partial x_N}(x) \\ \vdots & & \vdots \\ \frac{\partial f_N}{\partial x_1}(x) & \dots & \frac{\partial f_N}{\partial x_N}(x) \end{pmatrix}.$$

Simultaneous Nonlinear Equations

Convergence of Newton's method

The convergence analysis for $N > 1$ equations is exactly the same as that for $N = 1$ equations. The only complication is that the analysis involves second derivatives, and we have

$$f(x) \in \mathbb{R}^N, \qquad \nabla f(x) \in \mathbb{R}^{N \times N}, \qquad \nabla^2 f(x) \in \mathbb{R}^{N \times N \times N}.$$

Working with $\nabla^2 f(x)$ thus requires special notation for working with "cubes of numbers" which we do not have. I therefore omit the details.

The final result is analogous to that for $N = 1$: the error satisfies

$$\|x_{k+1} - x^\star\| = \mathcal{O}\big(\|x_k - x^\star\|^2\big)$$

if x_k is close enough to $x^\star$ and $\nabla f(x^\star)$ is not singular.

Simultaneous Nonlinear Equations

Broyden's method

Recap: the secant iteration is given by

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k).$$

Its main selling points are that it does not require derivatives, and it is faster than Newton's method if performance is measured in term of function evaluations.

Extending this method to $N > 1$ faces the challenge that the finite difference quotient

$$\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

does not make sense when x_k and $f(x_k)$ are vectors.

The underlying problem is that the first derivative of a function $f : \mathbb{R}^N \to \mathbb{R}^N$ is a matrix containing information about the slopes of all the components of $f(x)$ with respect to all the components of x, but comparing $f(x_k)$ and $f(x_{k-1})$ only informs us about the slopes in the direction $x_k - x_{k-1}$.

Simultaneous Nonlinear Equations

Broyden's method (continued)

The obstacle described above can be overcome by observing that the x_k should converge and hence x_k should be close to x_{k-1} and $\nabla f(x_k)$ should be close to $\nabla f(x_{k-1})$. Assuming we know $\nabla f(x_{k-1})$, it therefore makes sense to replace $\nabla f(x_k)$ with an approximation $\tilde{\nabla} f(x_k)$ determined by the following conditions.

- $\tilde{\nabla} f(x_k)\,(x_k - x_{k-1}) = f(x_k) - f(x_{k-1})$.
  Gradient is $\|f(x_k) - f(x_{k-1})\|_2 / \|(x_k - x_{k-1})\|_2$ in the $x_k - x_{k-1}$ direction.
- $\tilde{\nabla} f(x_k)\,\Delta x = \nabla f(x_{k-1})\,\Delta x$ for all $\Delta x \perp x_k - xk - 1$.
  Gradient remain unchanged in any other direction.

The unique $\tilde{\nabla} f(x_k)$ satisfying both of these constraints is

$$\tilde{\nabla} f(x_k) = \nabla f(x_{k-1}) + \frac{\Big(f(x_k) - f(x_{k-1}) - \nabla f(x_{k-1})\,(x_k - x_{k-1})\Big)(x_k - x_{k-1})^T}{\|x_k - x_{k-1}\|_2^2}.$$

Simultaneous Nonlinear Equations

Broyden's method (continued)

The update formula described above leads us to the following algorithm known as Broyden's method.

Algorithm 1 Broyden's method (slow version)

1: $\tilde{\nabla} f(x_0) = I$
2: **for** $k = 1, 2, 3, \ldots$ **do**
3: $\quad x_k = x_k - \tilde{\nabla} f(x_{k-1})^{-1} f(x_{k-1})$
4: $\quad \tilde{\nabla} f(x_k) = \tilde{\nabla} f(x_{k-1}) + \dfrac{\left(f(x_k) - f(x_{k-1}) - \tilde{\nabla} f(x_{k-1})(x_k - x_{k-1})\right)(x_k - x_{k-1})^T}{\|x_k - x_{k-1}\|_2^2}$
5: **end for**

Like the secant method, Broyden's method only requires a single function evaluation per iteration and no derivatives.

However, the above algorithm requires solving a potentially large and dense linear system $\tilde{\nabla} f(x_{k-1})^{-1} f(x_{k-1})$ in every iteration.

This is expensive but can be avoided using the result on the next slide.

Simultaneous Nonlinear Equations

Thm: Sherman-Morrison formula

Assume $A \in \mathbb{R}^{N \times N}$ is invertible, $x, y \in \mathbb{R}^N$ and $y^T A^{-1} x \neq -1$. Then, $A + xy^T$ is invertible and we have

$$(A + xy^T)^{-1} = A^{-1} - \frac{A^{-1} x y^T A^{-1}}{1 + y^T A^{-1} x}.$$

The operation $A + xy^T$ is called a rank-one update.

Proof. Check through straightforward calculations that

$$(A + xy^T)\left(A^{-1} - \frac{A^{-1} x y^T A^{-1}}{1 + y^T A^{-1} x}\right) = I.$$

See `https://en.wikipedia.org/wiki/Sherman-Morrison_formula` for details.

Simultaneous Nonlinear Equations

Broyden's method (continued)

We observe:

- The Broyden update for $\tilde{\nabla} f(x_k)$ is a rank-one update.
- We only need $\tilde{\nabla} f(x_k)^{-1}$ in Broyden's method, not $\tilde{\nabla} f(x_k)$.

Instead of keeping track of $\tilde{\nabla} f(x_k)$ and inverting it in every iteration, we can thus keep track of $\tilde{\nabla} f(x_k)^{-1}$ and use the Sherman-Morrison formula for the updates.

Working out the update formula for $\tilde{\nabla} f(x_k)^{-1}$ is tedious but straightforward. I omit the details and only present the final algorithm.

Algorithm 2 Broyden's method (fast version)

1: $\tilde{\nabla} f(x_0) = I$
2: **for** $k = 1, 2, 3, \ldots$ **do**
3: $\quad x_k = x_k - \tilde{\nabla} f(x_{k-1})^{-1} f(x_{k-1})$
4: $\quad \Delta x = x_k - x_{k-1}$
5: $\quad \Delta f = f(x_k) - f(x_{k-1})$
6: $\quad \tilde{\nabla} f(x_k)^{-1} = \tilde{\nabla} f(x_{k-1})^{-1} + \dfrac{\left(\Delta x - \tilde{\nabla} f(x_{k-1})^{-1}\right) \Delta x^T \tilde{\nabla} f(x_{k-1})^{-1}}{\Delta x^T \tilde{\nabla} f(x_{k-1})^{-1} \Delta f}$
7: **end for**

Simultaneous Nonlinear Equations

Implementation of Newton's and Broyden's methods

See `newton()`, `broyden()` and `convergence()`.

Observations:

- At best, Broyden converges in about twice as many iterations as Newton. This means that Broyden is effectively faster than Newton because every Broyden iteration runs in at most half the time of Newton's method.
- At worst, Broyden fails to converge even when Newton converges.

Simultaneous Nonlinear Equations

Gradient descent / steepest descent

Observation: solving a nonlinear equation is equivalent to solving an optimisation problem,

$$f(x^\star) = 0 \quad \Longleftrightarrow \quad x^\star = \arg\min_x \|f(x)\|_2^2.$$

The gradient descent method consists in repeatedly improving an initial guess x_0 by making small steps in the direction where $g(x) = \|f(x)\|_2^2$ decreases the fastest. This will take us closer and closer to a minimiser of $g(x)$ which is also a root of $f(x)$.

Recall from you calculus class that $\nabla g(x) \in \mathbb{R}^N$ is the direction where $g(x)$ increases the fastest. The steepest descent direction is thus given by $-\nabla g(x)$ and the gradient descent algorithm is as follows.

Algorithm 3 Gradient descent

1: **for** $k = 1, 2, 3, \ldots$ **do**

2: $\quad x_{k+1} = x_k - \alpha \nabla g(x_k)$

($\alpha > 0$ is a parameter of the algorithm. I will comment on this later.)

3: **end for**

Simultaneous Nonlinear Equations

Gradient descent (continued)

Implementing the gradient descent algorithm requires a formula for $\nabla g(x)$. We observe:

$$g(x) = \sum_{i=1}^{N} f_i(x)^2 \qquad \longrightarrow \qquad \nabla g(x) = 2 \sum_{i=1}^{N} f_i(x) \nabla f_i(x).$$

Rewriting the formula on the right-hand side in matrix notation yields

$$\nabla g(x) = \nabla f(x)^T f(x).$$

See `gradient_descent()` and `convergence()`.

We observe:

- Gradient descent diverges if α is too large.
- Gradient descent converges very slowly if α is too small.

Somewhere between these two extremes must be an α which leads to the fastest possible convergence. This optimal α is different for different $g(x)$ and may even be different for different x.

Simultaneous Nonlinear Equations

Choice of step size α

We conclude from the above that α should ideally be chosen adaptively depending on $g(x)$ and the current position x_k.

Two common ways for doing so are:

- Choose $\alpha = \arg\min_{\alpha>0} g\big(x_k - \alpha\, \nabla g(x_k)\big)$.
- Start with $\alpha_0 = 1$ and set $\alpha_\ell = \alpha_{\ell-1}/2$ until we reach an α_ℓ such that $g\big(x_k - \alpha_\ell \nabla g(x_k)\big) < g(x_k)$.

The first approach ensures that we decrease $g(x_{k+1})$ as much as possible, but the resulting one-dimensional optimisation problem may be hard to solve.

Both approaches ensure that $g(x_{k+1}) < g(x_k)$; thus they guarantee that gradient descent always gets closer to the root.

This is the main selling point of gradient descent: it has a much lower risk of failing compared to e.g. Newton's method.

Simultaneous Nonlinear Equations

Remark

Using gradient descent for solving nonlinear equations corresponds to traversing the equivalence

$$f(x^\star) = 0 \quad \Longleftrightarrow \quad x^\star = \arg\min_x \|f(x)\|_2^2$$

from left to right.

The other direction is often the reason why we are interested in solving nonlinear equations in the first place: we want to find the stationary points of a function $g : \mathbb{R}^N \to \mathbb{R}$, i.e. the points $x^\star$ such that $\nabla g(x^\star) = 0$.

Simultaneous Nonlinear Equations

Remark

Gradient descent is very popular in machine learning.

The problem there is that we have data $(x_i, y_i)_{i=1}^N$ and an ansatz $f(x, \theta)$ involving some parameters θ, and we would like to find the θ which reproduces the data as accurately as possible.

Example: x_i are pictures of pets, and $y_i = [1, 0]$ if x_i shows a cat and $y_i = [0, 1]$ if x_i shows a dog. We then train a neural network $f(x, \theta)$ which given a picture x outputs the likelihood that the picture is a cat or a dog.

Formally, the machine learning problem is to find

$$\theta = \arg\min_{\theta} \sum_{i=1}^{N} \left| f(x_i, \theta) - y_i \right|^2.$$

Gradient descent is well-suited for this problem because its runtime per iteration scales linearly in N which allows us to process very large data sets.

Simultaneous Nonlinear Equations

Remark

Recall the conjugate gradients equations

$$x_\ell = x_{\ell-1} + \alpha_\ell\, p_{\ell-1}, \qquad r_\ell = r_{\ell-1} - \alpha_\ell\, Ap_{\ell-1}, \qquad p_\ell = r_\ell + \beta_\ell\, p_{\ell-1}$$

and observe that $r_\ell = b - Ax_\ell$ since we start with $x_0 = 0$ and $r_0 = b$.

One can show

$$g(x) = \|x - A^{-1}b\|_A^2 \quad \Longrightarrow \quad \nabla g(x) = Ax - b = -r.$$

Thus if we set $\beta_\ell = 0$, then $p_\ell = r_\ell = -\nabla g(x_\ell)$ and conjugate gradients is exactly gradient descent applied to the $g(x)$ given above.

Introducing β_ℓ serves to ensure that the descent directions p_ℓ are mutually conjugate, i.e. they satisfy

$$p_k^T A p_\ell = \begin{cases} 1 & \text{if } k = \ell, \\ 0 & \text{otherwise.} \end{cases}$$

This explains why the method is called *conjugate gradients*.

Simultaneous Nonlinear Equations

Summary

- Bracketing methods cannot be generalised to $N > 1$ equations. This implies that there is no algorithm which is guaranteed to find a root for $N > 1$ equations.
- Newton's method is virtually the same for $N = 1$ and $N > 1$.
- Broyden's method generalises the secant method to $N > 1$ equations.
- Finding $x^\star$ such that $f(x^\star) = 0$ is equivalent to solving the optimisation problem $\min_x g(x)$ where $g(x) = \|f(x)\|_2^2$.

  The gradient descent method consists in repeatedly making small steps in the direction where $g(x)$ decreases the fastest.

  Gradient descent also requires derivatives, but it is typically more robust with respect to the initial guess.