Lecture 11, Sci. Comp. for DPhil Students

Nick Trefethen, Tuesday 22.11.18

Today

- III.3 Newton's method for minimizing a function of one variable
- III.4 Newton's method for minimizing a function of several variables
- III.5 From Newton's method to practical optimization

Handouts

- m22_pureNewtonmin.m and m23_fminunc.m, m23b.m, m23c.m, m23d.m
- Table of contents of Nocedal and Wright, Numerical Optimization

Announcements

Recall last lecture, where we began optimization

III. OPTIMIZATION

III.1 Newton's method for a single equation
III.2 Newton's method for a system of equations

F(x) - vector function to be zeroed

III.3 Newton's method for minimising a function of one variable

Given: function f(x).

Goal: find a **minimum** x^* s.t. $f(x^*) = \min$.

Global minimum: typically hard or impossible to be sure.

So instead we usually seek a **local minimum**:

 $f(x) \ge f(x^*)$ for all x in a neighbourhood of x^* .

Obvious idea: use Newton's method to solve f'(x) = 0.

Newton's method (pure and impractical in this form!)

Equivalent formulation:

- (1) Approximate f(x) near x_k by a parabola
- (2) Set $x_{k+1} = \text{minimum of this parabola (or maximum?!)}$

[Draw a sketch.]

III.4 Newton's method for minimizing a function of several variables

Consider now $f: \mathbb{R}^n \to \mathbb{R}$. Seek $x^* \in \mathbb{R}^n$ s.t. $f(x^*) = \text{local minimum}$, i.e.,

$$f(x_1, \ldots, x_n) = \text{local minimum}.$$

Of course there's an analogous Newton method to what we had for a system of equations. But now we'll need second derivatives.

Given $f: \mathbb{R}^n \to \mathbb{R}$, the **gradient** of f at $x \in \mathbb{R}^n$ is the n-vector

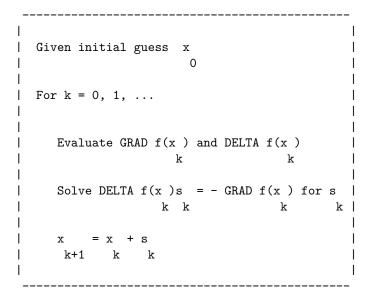
$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

The **Hessian** of f at $x \in \mathbb{R}^n$ is the symmetric $n \times n$ matrix

$$\Delta f(x) = [\nabla f]'(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Notice that the Hessian is the Jacobian of the gradient.

Newton's method



Note how this relates to last lecture's Newton method for zero finding for a system:

Given
$$F: \mathbb{R}^n \to \mathbb{R}^n$$
, seek $x^* \in \mathbb{R}^n$ s.t. $F(x^*) = 0$.

Note also that the linear algebra at each step involves a symmetric matrix.

| k+1 k k -----

In MATLAB: the far more robust codes than these pure Newton ideas are fminbnd for a scalar problem, fminunc for a system.

With pure Newton, at points where the objective function is not convex, the Newton step isn't even a descent direction.

[m22_pureNewtonmin.m] starting from e.g. (2,1), (2,2), (2,-2)

III.5 From Newton's method to practical optimization

Look at Nocedal & Wright table of contents, e.g. chaps. 3, 4, 7, 8, 9

A fundamental observation

Newton's method is 2nd-order accurate, and this may seem somewhat arbitrary. It sounds better than 1st-order, worse than 3rd-order,....

The reality is different. All algorithms of order > 1 are equivalent up to constant factors. (E.G., 2 steps of Newton has 4th order.) Thus Newton really is special: the simplest superlinear method.

Limitations to Newton's method

1. Speed

- (a) How to compute all the necessary derivatives? These days, often with automatic differentiation: see book by Griewank and Walther and also www.autodiff.org.
- (b) How to solve the large linear algebra problems repeatedly? These days, often with CG and related iterative methods.

Practical algorithms settle for inexact derivatives: **inexact Newton** Also, for both (a) and (b), sparsity is invaluable (and common).

2. Robustness

Pure Newton iteration is nearly useless in practice, and a long way from software. For making algorithms more robust here are some big ideas:

(a) Modified Newton

Instead of the true Hessian H, use e.g. H + aI for some a. This is the flavor of a true citation classis: a 1963 SIAM paper by Marquardt for which Google Scholar lists 30,920 citations! (The details are more complicated.)

(b) Line searches

Instead of $x_{k+1} = x_k + s_k$, use $x_{k+1} = x_k + a_k s_k$, where the **step length** a_k is chosen small enough to guarantee monotonic descent towards solution.

(c) Trust regions

A different concept, but similar efficacy to line searches in practice.

We now demonstrate fminunc with a code to illustrate how it estimates derivatives and takes more cautious steps than a pure Newton method.

Here is a harder example, which has been a test example for decades: the Rosenbrock function,

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2.$$

We also modify the call to fminunc to use gradients as well as function values.

[m23b.m]

Here is a harder variant of the Rosenbrock function, which we show in more of a movie mode:

$$f(x,y) = (1-x)^2 + 100(y + \cos(\pi x))^2.$$

[m23c.m] (starting e.g. from (-2.5, 2))

Chebfun2 does well at this problem, using its methods of global optimization (which are however restricted to 1D and 2D).

[m23d.m]