Algorithms for optimization

Finding derivatives, unconstrained and constrained optimization

Zdeněk Hurák

Department of Control Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague

March 2, 2021

If possible, we want to use them (why shouldn't it be possible?).

How to compute them?

- symbolically
- numerically (finite difference, FD)
- algorithmic (also automatic) differentiation (AD)

Sometimes symbolic differentiation not available/usable

Why?

Finite difference (FD) computation of derivatives

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \approx \frac{f(x+\alpha) - f(x)}{\alpha}$$

forward difference

or

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \approx \frac{f(x) - f(x-\alpha)}{\alpha}$$

backward difference

or

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \approx \frac{f(x+\frac{\alpha}{2}) - f(x-\frac{\alpha}{2})}{\alpha}$$

central difference

Troubles with FD?

Algorithmic (also automatic) differentiation (AD)

Systematic application of chain rule for derivatives of composed functions.
How does it differ from symbolic and how from numerical differentiation?
Two versions: forward and reverse AD.

Implementation of forward AD using dual numbers

Similar to complex numbers, a dual number has two components:

$$x = v + d\epsilon, \quad \epsilon^2 = 0$$

Multiplication of two dual numbers $y = x_1 \cdot x_2$

$$\begin{aligned} x &= (v_1 + d_1\epsilon) \cdot (v_2 + d_2\epsilon) \\ &= v_1 v_2 + (v_1 d_2 + d_1 v_2)\epsilon \end{aligned}$$

Implementations of AD in various languages

Matlab: recently added to Optimization Toolbox for Matlab, CasADi, MAD (MatlabAD) from Tomlab, . . .

Python: CasADi, . . .

Julia: ForwardDiff.jl, ReverseDiff, Zygote.jl, . . .

- descent direction methods
- trust region methods

- finding a descent direction
- line search

$$\boxed{x_{k+1} = x_k + \alpha_k d_k}$$

Descent direction

Directional derivative negative

$$\nabla f(\mathsf{x}_k)^{\mathrm{T}}\mathsf{d}_k < 0$$

But beware of higher order terms – descent direction condition valid only in some vicinity of x.

Steepest descent (aka gradient) method

$$\mathsf{d}_k = -\nabla f(\mathsf{x}_k)$$

$$\boxed{\mathsf{x}_{k+1} = \mathsf{x}_k - \alpha_k \nabla f(\mathsf{x}_k)}$$

1. fixed step
2. exact search
3. approximate search

L-smoothness (Lipschitz continuity of the gradient)

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$$

If second derivative exists, L is an upper bound

$$\|\nabla^2 f\| \leq L$$

For quadratic functions

$$L = \max \lambda_i(\mathrm{Q})$$

Quadratic dominance (descent lemma)

$$f(\mathrm{x}_{k+1}) \leq f(\mathrm{x}_k) + \nabla f(\mathrm{x}_k)^\mathsf{T}(\mathrm{x}_{k-1} - \mathrm{x}_k) + \frac{L}{2}\|\mathrm{x}_{k-1} - \mathrm{x}_k\|^2$$

Step length

$$\boxed{\alpha = \frac{1}{L}}$$

Exact line search

Several methods (bisection, golden section, Newton, ...)
For quadratic functions $f(\mathrm{x}) = \frac{1}{2}\mathrm{x}^\mathsf{T}\mathrm{Q}\mathrm{x} + \mathrm{c}^\mathsf{T}\mathrm{x}$ closed-form formula.

$$\underset{\alpha_k}{\text{minimize}}\, f(\mathrm{x}_k + \alpha_k \mathrm{d}_k)$$

$$\begin{aligned} f(\mathrm{x}_k + \alpha_k \mathrm{d}_k) &= \frac{1}{2}(\mathrm{x}_k + \alpha_k \mathrm{d}_k)^\mathsf{T}\mathrm{Q}(\mathrm{x}_k + \alpha_k \mathrm{d}_k) + \mathrm{c}^\mathsf{T}(\mathrm{x}_k + \alpha_k \mathrm{d}_k) \\ &= \frac{1}{2}\mathrm{x}_k^\mathsf{T}\mathrm{Q}\mathrm{x}_k + \mathrm{d}_k^\mathsf{T}\mathrm{Q}\mathrm{x}_k\alpha_k + \frac{1}{2}\mathrm{d}_k^\mathsf{T}\mathrm{Q}\mathrm{d}_k\alpha_k^2 + \mathrm{c}^\mathsf{T}(\mathrm{x}_k + \alpha_k \mathrm{d}_k) \end{aligned}$$

$$\frac{\mathrm{d}f(\mathrm{x}_k + \alpha_k \mathrm{d}_k)}{\mathrm{d}\alpha_k} = \mathrm{d}_k^\mathsf{T}(\mathrm{Q}\mathrm{x}_k + \mathrm{c}) + \mathrm{d}_k^\mathsf{T}\mathrm{Q}\mathrm{d}_k\alpha_k = 0$$

$$\boxed{\alpha_k = -\frac{\mathrm{d}_k^\mathsf{T}(\mathrm{Q}\mathrm{x}_k + \mathrm{c})}{\mathrm{d}_k^\mathsf{T}\mathrm{Q}\mathrm{d}_k} = -\frac{\mathrm{d}_k^\mathsf{T}\nabla f(\mathrm{x}_k)}{\mathrm{d}_k^\mathsf{T}\mathrm{Q}\mathrm{d}_k}}$$

Approximate line search – backtracking

Usually the exact minimum not needed, *sufficient descent* is enough.

Armijo / Wolfe conditions.

Backtracking algorithm: parameters s, $\beta \in (0, 1)$, $\gamma \in (0, 1)$:
Set $\alpha_k = s$
While

$$\boxed{f(\mathsf{x}_k) - f(\mathsf{x}_k + \alpha_k \mathsf{d}_k) < -\gamma \alpha_k \mathsf{d}^T \nabla f(\mathsf{x}_k),}$$

set

$$\boxed{\alpha_k = \beta \alpha_k.}$$

When to stop the iterations?

Gradient method through examples

Scaled gradient method for ill-conditioned problems

$$\underset{x \in \mathbb{R}^2}{\text{minimize}}\, x^T Q x,$$

where the matrix Q is

$$Q = \begin{bmatrix} 1000 & 20 \\ 20 & 1 \end{bmatrix}.$$

Condition number κ for a given matrix A is

$$\kappa(\mathsf{A}) = \|\mathsf{A}^{-1}\| \cdot \|\mathsf{A}\|.$$

It can be computed as ratio of the largest and smallest singular values, that is,

$$\kappa(\mathsf{A}) = \frac{\sigma_{\mathsf{max}}(\mathsf{A})}{\sigma_{\mathsf{min}}(\mathsf{A})}.$$

Ideally should be around 1.
In the example above is well above 1000.

Scaling to improve the conditioning and the convergence

Introduce new variable variable y

$$\mathsf{x} = \mathsf{S}\mathsf{y},$$

The optimization cost changes $f(\mathsf{Sy})$. Relabel it to $g(\mathsf{y})$.
Chain rule

$$\nabla g(\mathsf{y}) = \mathsf{S}^T \nabla f(\mathsf{Sy}).$$

Steepest descent iterations then change accordingly

$$\mathsf{y}_{k+1} = \mathsf{y}_k - \alpha_k \nabla g(\mathsf{y}_k)$$

$$\mathsf{y}_{k+1} = \mathsf{y}_k - \alpha_k \mathsf{S}^T \nabla f(\mathsf{S}\mathsf{y}_k)$$

$$\underbrace{\mathsf{S}\mathsf{y}_{k+1}}_{\mathsf{x}_{k+1}} = \underbrace{\mathsf{S}\mathsf{y}_k}_{\mathsf{x}_k} - \alpha_k \underbrace{\mathsf{S}\mathsf{S}^T}_{\mathsf{D}} \nabla f(\underbrace{\mathsf{S}\mathsf{y}_k}_{\mathsf{x}_k})$$

Defining the scaling matrix D as SS^T, a single iteration changes to

$$\boxed{\mathsf{x}_{k+1} = \mathsf{x}_k - \alpha_k \mathsf{D}_k \nabla f(\mathsf{x}_k).}$$

How to choose the scaling matrix?

Make the Hessian matrix $\nabla^2 f(\mathsf{Sy})$ (the matrix Q above) better conditioned. Ideally, $\nabla^2 f(\mathsf{Sy}) \approx \mathsf{I}$.
Chain rule once again

$$\begin{aligned}\nabla^2 g(\mathsf{y}) &= \mathsf{S}^T \nabla^2 f(\mathsf{Sy}) \mathsf{S} \\ &= \mathsf{D}^{\frac{1}{2}} \nabla^2 f \mathsf{D}^{\frac{1}{2}}\end{aligned}$$

A simple way using a diagonal scaling matrix D

$$\mathsf{D}_{ii} = [\nabla^2 f(\mathsf{x}_k)]_{ii}^{-1}.$$

Newton's method for solving eqautions

Solve

$$g(x) = 0.$$

Approximate g at x_k using a linear function

$$\underbrace{g(x_{k+1})}_{0} = g(x_k) + g'(x_k)(x_{k+1} - x_k)$$

$$0 = g(x_k) + g'(x_k)x_{k+1} - g'(x_k)x_k,$$

from which the famous formula follows

$$\boxed{x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}.}$$

In the vector case

$$\boxed{\mathrm{x}_{k+1} = \mathrm{x}_k - \mathrm{J}(\mathrm{x}_k)^{-1}\mathrm{g}(\mathrm{x}_k).}$$

$$\underset{x \in \mathbb{R}}{\text{minimize}} \quad f(x)$$

Model the function f at x_k using a quadratic function $m_k(x)$

$$m_k(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(x_k)(x - x_k)^2.$$

At the k-th iteration

$$\underset{x_{k+1} \in \mathbb{R}}{\text{minimize}} \quad m(x_{k+1})$$

Straightforward: find the value of x_{k+1} for which the derivative of $m_k()$ vanishes.

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

Full vector version

$$\boxed{\mathrm{x}_{k+1} = \mathrm{x}_k - (\nabla^2 f(\mathrm{x}_k))^{-1} \nabla f(\mathrm{x}_k).}$$

Discussion of Newton's method

$$\mathrm{x}_{k+1} = \mathrm{x}_k - \alpha_k \nabla^2 f(\mathrm{x}_k)^{-1} \nabla f(\mathrm{x}_k).$$

Generalization of the (scalar) secant method.

Secant approximation of the derivative (for rootfinding)

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

$$x_{k+1} = x_k - \underbrace{\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}}_{\approx \dot{f}(x_k)} f(x_k)$$

Secant approximation of the derivative (for optimization)

$$\ddot{f}(x_k) \approx \frac{\dot{f}(x_k) - \dot{f}(x_{k-1})}{x_k - x_{k-1}} =: b_k$$

$$b_k(\underbrace{x_k - x_{k-1}}_{s_{k-1}}) = \underbrace{\dot{f}(x_k) - \dot{f}(x_{k-1})}_{y_{k-1}} \qquad \text{secant condition}$$

Secand condition in the vector case

$$\boxed{B_{k+1}s_k = y_k}$$

B_k is a matrix with Hessian-like properties

$$B_k = B_k^T$$

$$B_k \succ 0$$

How to get it? Updates.

$$B_{k+1} = B_k + \text{some ``small'' update}$$

Possibly updating B_{k+1}^{-1} directly. One popular update is BFGS:

$$\boxed{H_{k+1} = H_k + \left(1 + \frac{y_k^T H_k y_k}{s_k^T y_k}\right) \cdot \frac{s_k s_k^T}{s_k^T y_k} - \frac{s_k y_k^T H_k + H_k y_k s_k^T}{y_k^T s_k}}$$

Approximate $f()$ at x_k with some model $m_k()$, typically a quadratic function

$$m_k(\mathsf{p}) = f(\mathsf{x}_k) + \nabla f(\mathsf{x}_k)^\mathsf{T}\mathsf{p} + \frac{1}{2}\mathsf{p}^\mathsf{T}\underbrace{\nabla^2 f(\mathsf{x}_k)}_{\text{or}\approx}\mathsf{p}$$

but trust the model only within

$$\|\mathsf{p}\|_2 \leq \delta$$

$$\boxed{\begin{aligned}&\underset{\mathsf{p}\in\mathbb{R}^n}{\text{minimize}}\ m_k(\mathsf{p})\\ &\text{subject to } \|\mathsf{p}\|_2 \leq \delta\end{aligned}}$$

Evaluating the predictive performance of the model

and shrinking or expanding the trust region. Use

$$\eta = \frac{\text{actual improvement}}{\text{predicted improvement}} = \frac{f(x_k) - f(x_{k+1})}{f(x_k) - m_k(x_{k+1})}$$

Shrink for small η (≈ 0) and expand for larger η (≈ 1).

Projected gradient method