Steepest descent

needs help

Ed Bueler

MATH 661 Optimization

Fall 2022

steepest descent for unconstrained optimization

- these slides are a brief introduction to a well-known topic in unconstrained optimization
- steepest descent
  - a.k.a. gradient descent
- please read sections 12.1 and 12.2 of the textbook[1]
  - ignore the Lemmas for now

[1] Griva, Nash & Sofer, *Linear and Nonlinear Optimization*, 2nd ed., SIAM Press 2009

why you should know about steepest descent

- widely used for optimization in the "real world"
- for easy problems it is the lazy-person's algorithm
  - "easy" roughly means:
    - smooth
    - dimension $< 10^6$ (or so)
    - unconstrained
  - I don't recommend steepest descent, but I acknowledge it might minimize total programmer time
- for hard problems it may be the only thing you can implement
  - e.g. big machine learning problems, big nonlinear inverse problems, ...
  - sometimes as *stochastic gradient descent* (popular buzzword!)
    - it's even slower (worse?) than ordinary steepest descent
  - a version of steepest descent may be the standard in your industry

the steepest descent algorithm

- assume $f : \mathbb{R}^n \to \mathbb{R}$ has (at least) one continuous derivative
- we want to solve the unconstrained problem:

$$\min_{\mathbb{R}^n} f(x)$$

- the steepest descent algorithm:

  1. User supplies x_0.
  2. For $k = 0, 1, 2, \dots$
     (i) If x_k is optimal then stop.
     (ii) Search direction is
     $$p_k = -\nabla f(x_k)$$
     (iii) Determine step length $\alpha_k > 0$. Let $x_{k+1} = x_k + \alpha_k p_k$.

steepest descent is obvious

- steepest descent is an obvious interpretation of "'General Optimization Algorithm II" in §2.4
  - direction is chosen as "go straight downhill"
    - the gradient points straight uphill
  - but we don't know how to use the length of $\nabla f(x_k)$
  - so we *must* make a nontrivial step-length choice for α_k
  - also we need a stopping criterion
- *any* choice of steepest descent length, i.e. $p_k = -c\nabla f(x_k)$ and $c > 0$, generates a (feasible) descent direction at x_k
  - recall: p is a *descent direction at x* if $p^\top \nabla f(x) < 0$
- *fun fact:* the direction of $p_k = -\nabla f(x_k)$ solves this optimization problem

$$\min_{\|q\|=1} q^\top \nabla f(x_k)$$

one way to choose step length: back-tracking

- we will see in section 11.5 that we can prove convergence of many unconstrained optimization algorithms as long as the step-size α_k is chosen to satisfy certain conditions
  - this is the line search idea
- for now I just need *some* reasonable way to choose α_k
- the most common way to satisfy these conditions is “back-tracking”
  - page 378 of the textbook
  - an implementation:

```
function alpha = bt(xk,pk,dfxk,f)
Dk = dfxk' * pk;      % scalar directional derivative; negative
c = 1.0e-4;           % modest sufficient decrease
rho = 0.5;            % backtracking by halving
alpha = 1.0;
while f(xk + alpha * pk) > f(xk) + c * alpha * Dk
    alpha = rho * alpha;
end
```

- we will return to this topic, and prove remarkable Theorem 11.7

steepest-descent with back-tracking code

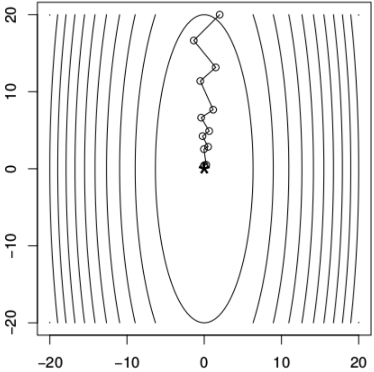
- here is a basic implementation of *steepest-descent* with *back-tracking*
  = SDBT
- it assumes that the user supplies x_0 and a function f that returns both the values $f(x)$ and the gradient $\nabla f(x)$:

```
xk = x0;
for k = 1:maxiters
    [fxk, dfxk] = f(xk);
    if norm(dfxk) < tol
        break                          % and report success
    end
    pk = - dfxk;                       % steepest descent
    alpha = bt(xk,pk,dfxk,f);          % back-tracking
    xk = xk + alpha * pk;
end
```

- set `maxiters` to 10^4 or so to avoid long waits for failure

steepest-descent-back-tracking: example I

- suppose $f(x) = 5x_1^2 + \frac{1}{2}x_2^2$ for $x \in \mathbb{R}^2$, an easy quadratic objective function with global minimum at $x^* = (0,0)^\top$
- result from SDBT:



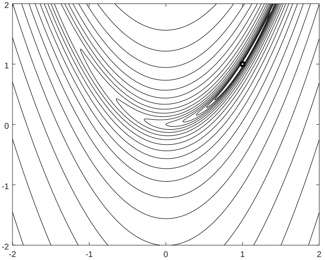
- is this result o.k.?

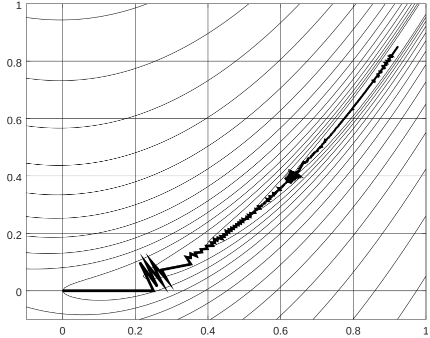
- a famously-harder problem in $\mathbb{R}^2$ is to minimize the *Rosenbrock function*:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

  - a quartic polynomial in 2 variables
  - has a single global minimum at $x^* = (1, 1)^\top$
  - has steep "banana" shaped contours (bottom left)

- at right: SDBT from $x_0 = (0, 0)^\top$
  - struggles





quadratic functions

- consider general quadratic functions in $\mathbb{R}^n$
- such functions can always be written

$$f(x) = \frac{1}{2} x^\top Q x - c^\top x + d$$

  - Q is a symmetric square matrix, c is a column vector, $d \in \mathbb{R}$
  - recall that
$$\nabla f(x) = Q x - c$$
  - example I above: $c = 0$ and $Q = \begin{bmatrix} 5 & 0 \\ 0 & 1/2 \end{bmatrix}$
- if Q is positive definite then
  - f is strictly convex, and
  - there is unique global minimizer where $\nabla f = 0$: $\qquad x^* = Q^{-1} c$

line search for quadratic functions

- given any descent direction p_k at x_k, the *optimal* step size is

$$\alpha_k = \frac{-p_k^\top \nabla f(x_k)}{p_k^\top Q p_k} = \frac{p_k^\top (c - Q x_k)}{p_k^\top Q p_k}$$

  - Exercise **P13** on Assignment # 7
- this α_k minimizes $g(\alpha) = f(x_k + \alpha p_k)$ over $\alpha > 0$
- thus back-tracking is *not* needed for quadratic functions

- but steepest descent is still slow
  - Exercise **P14** asks you to reproduce Example 12.1 in section 12.2 of the textbook, in which steepest descent with optimal step size uses a totally-unnecessary 216 steps to get modest accuracy
  - fundamentally, the steepest descent direction is wrong

steepest descent is the wrong direction

- for quadratic objective functions $f(x) = \frac{1}{2}x^\top Qx - c^\top x$,
  the Newton iteration converges to $x^* = Q^{-1}c$ in one step

- Newton uses this direction:

$$p_k = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$$

- steepest descent uses:

$$p_k = -I^{-1}\nabla f(x_k)$$

- the identity I is the wrong matrix; it should be the Hessian of f at x_k

- unconstrained optimization needs the information in the Hessian $\nabla^2 f(x_k)$, which rotates and scales the steepest descent vector $-\nabla f(x_k)$ to be an accurate step toward the minimum
  - that's why it is worth reading Chapters 11, 12, and 13!
  - especially "quasi-Newton" methods
  - however, computing and inverting the Hessian is expensive!

summary

- steepest descent simply uses search direction $p_k = -\nabla f(x_k)$
- determining the step size α_k, when actually taking the step $x_{k+1} = x_k + \alpha_k p_k$, is nontrivial
  - line search (section 11.5) or trust region (11.6) is needed
  - for general functions, back-tracking is reasonable
  - for quadratic functions we can use the optimal step size
- even with good line search, steepest descent sucks
  - steepest descent is slow when contour lines (level sets) are highly curved
  - going down the gradient is generally the wrong direction:
    - steepest descent direction $p_k = -I^{-1}\nabla f(x_k)$ is wrong, while
    - Newton step direction $p_k = -(\nabla^2 f(x_k))^{-1}\nabla f(x_k)$ is perfect for quadratic optimization
  - the steepest-descent vector $p_k = -\nabla f(x_k)$ has a length which depends on the scaling of $f(x)$, which is bad; the Newton step does not have this flaw
- unfortunately, functions like Rosenbrock remain difficult even for Newton