Steepest descent

needs help

Ed Bueler

MATH 661 Optimization

Fall 2024

steepest descent for unconstrained optimization

- these slides are a brief introduction to a well-known topic in unconstrained optimization: steepest descent
  - also known as gradient descent
- please read sections 12.1 and 12.2 of the textbook,[1] but just ignore the Lemmas for now; we will get back to it
- codes seen in these slides are already posted at the Codes tab of the public site

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

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) User provides stopping criterion: If x_k is optimal then stop.
   (ii) Search direction is $p_k = -\nabla f(x_k)$.
   (iii) User determines step length $\alpha_k > 0$.
   (iii) 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
  - and an initial iterate

- *any* choice of steepest descent length ($p_k = -\alpha \nabla f(x_k)$ and $\alpha > 0$) 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:* if $\nabla f(x_k) \neq 0$ then 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
mu = 1.0e-4;          % modest sufficient decrease
rho = 0.5;            % backtracking by halving
alpha = 1.0;
while f(xk + alpha * pk) > f(xk) + mu * 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 $\nabla f(x)$:

```
function [z, xk, k] = sdbt(f, x0, tol)

xk = x0(:);
maxiters = 10000;
for k = 1:maxiters
    [fk, dfk] = f(xk);              % objective and gradient
    if norm(dfk) < tol
        z = fk;
        break                       % success
    end
    pk = - dfk(:);                  % steepest descent
    alpha = bt(xk, pk, dfk, f);    % back-tracking
    xk = xk + alpha * pk;
end
```

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 $x^* = (0,0)^\top$
- using the codes:

```
>> x0 = [2 20]';    % start far away
>> [z, xk, k] = sdbt(@easyq, x0, 1.0e-10)
z = 1.8601e-21
xk =
   3.6456e-12
   5.9894e-11
k =   105
```

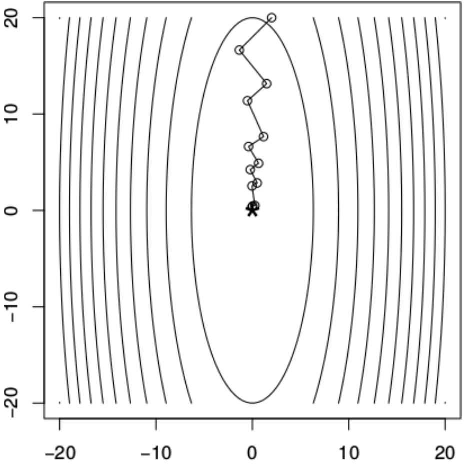
- this is based on a function which returns $f(x)$ and $\nabla f(x)$:

```
function [fx, dfx] = easyq(x)

fx = 5 * x(1)^2 + 0.5 * x(2)^2;
dfx = [10 * x(1);
       x(2)];
```

steepest-descent-back-tracking: example I

- recall: $f(x) = 5x_1^2 + \frac{1}{2}x_2^2$, $x_0 = (2, 20)^\top$
- result from SDBT:



- is this result o.k.?

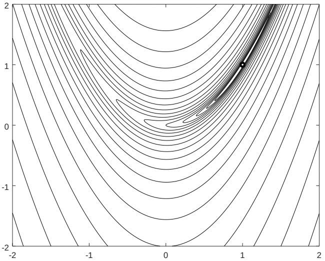
steepest-descent-back-tracking: example II

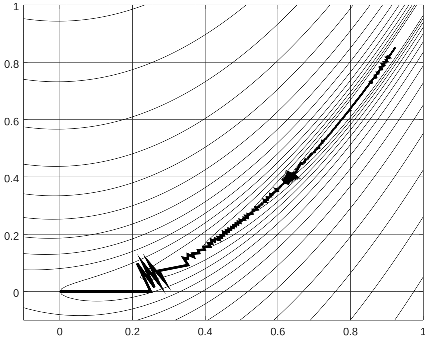
- a famously-harder problem in $\mathbb{R}^2$ is to minimize the *Rosenbrock function*:

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

  - 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
$$\nabla f(x) = Qx - c$$
  - example I above: $c = 0$, $d = 0$, 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 $\nabla f = 0$: $\qquad x^* = Q^{-1} c$
- so quadratic functions are easy to handle, but steepest descent does a bad job!

line search for quadratic functions

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

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

  - Exercise **P15** 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 **P16** asks you to reproduce Example 12.1 in section 12.2 of the textbook. In this example, steepest descent with optimal step size uses a totally-unnecessary 216 steps to get modest accuracy.
  - fundamentally, the steepest descent direction is wrong,
  - and the steepest descent idea by itself does not address how far to go

line search for quadratic functions

- given any descent direction p_k at x_k, for quadratic functions 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 **P15** 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 **P16** asks you to reproduce Example 12.1 in section 12.2 of the textbook. In this example, steepest descent with optimal step size uses a totally-unnecessary 216 steps to get modest accuracy.
  - fundamentally, the steepest descent direction is wrong,
  - and the steepest descent idea by itself does not address how far to go

steepest descent is the wrong direction

- for quadratic objective functions $f(x) = \frac{1}{2}x^\top Q x - c^\top x$,
  the Newton iteration converges to $x^* = Q^{-1}c$ in one step
- Newton uses this search direction p_k which solves:

$$\nabla^2 f(x_k)\, p_k = -\nabla f(x_k)$$

- steepest descent uses p_k which solves:

$$I\, p_k = -\nabla f(x_k)$$

- the identity I is the wrong matrix; for quadratic functions 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 solving with the Hessian is expensive

summary

- steepest descent (gradient descent) simply uses the search direction $p_k = -\nabla f(x_k)$
- determining the step size α_k, when actually taking the step, namely $x_{k+1} = x_k + \alpha_k p_k$, 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 direction $p_k = -(\nabla^2 f(x_k))^{-1}\nabla f(x_k)$ is perfect for quadratic objectives
  - 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
- however, functions like Rosenbrock remain difficult even for Newton

the other thing you should know ...

- for machine learning (ML) problems, a version called *stochastic gradient descent* (SGD) is the industry baseline
  - Adam, etc. are based on SGD, but with added “moment tracking”
- if ML were actually optimization, as it is usually portrayed, this would be very odd
- ... however, ML is not really optimization, but a different game
- I hope to tell the story by the end of the semester