steepest descent, Newton method, and back-tracking line search: demonstrations

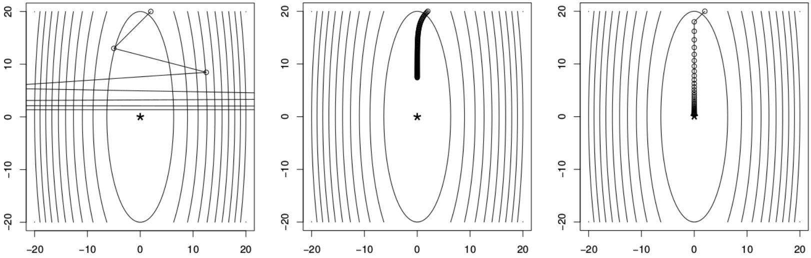
Ed Bueler

Math 661 Optimization

September 26, 2016

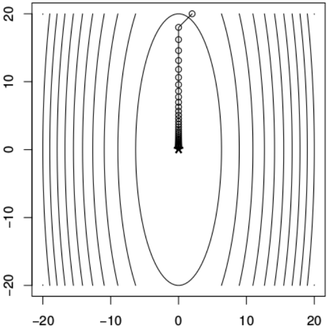
steepest descent with fixed steps

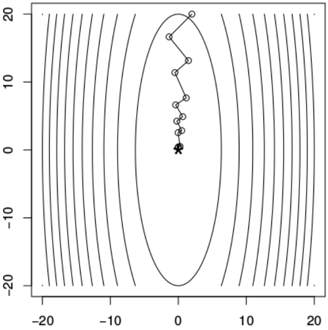
- consider the function $f(x) = 5x_1^2 + \frac{1}{2}x_2^2$
- try steepest descent: $p_k = -\nabla f(x_k)$, $x_{k+1} = x_k + \alpha_k p_k$
- fixed α_k: can get overshoot (*left*) or many small steps (*middle*)
- ...one might "hand-tune" steps for reasonable number (*right*)



steepest descent: backtracking seems to help

- "hand-tuned" (*left*) and back-tracking (*right*) results seem to be comparable in number of steps
  - more on back-tracking soon
- are either good?
- ...remember that for *this* function, which is quadratic ($f(x) = \frac{1}{2}x^\top Qx$), the Newton method converges in one step



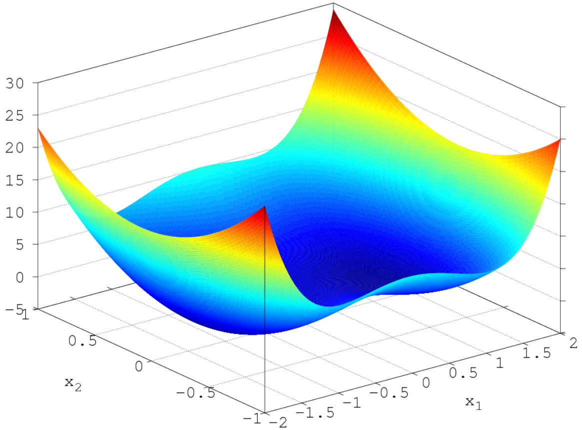


more interesting example function

- consider the problem "$\min_{x \in \mathbb{R}^2} f(x)$" for this function:

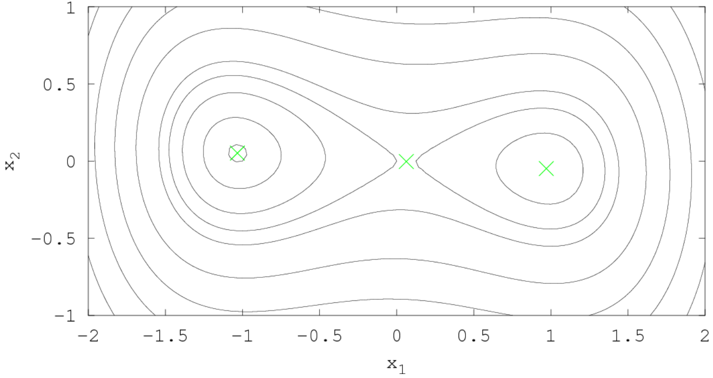
$$f(x) = 2x_1^4 - 4x_1^2 + 10x_2^2 + \frac{1}{2}x_1 + x_1 x_2$$

  - quartic, but not "hard" like Rosenbrock
  - visualized as a surface:



3 stationary points, 2 local min, 1 global min

- a clearer visualization as contours
- recall "stationary point" means $\nabla f(x) = 0$ (green $\times$)

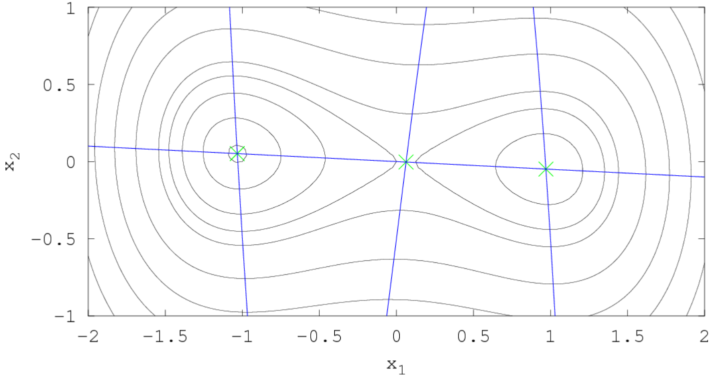


visualize equations $\nabla f(x) = 0$

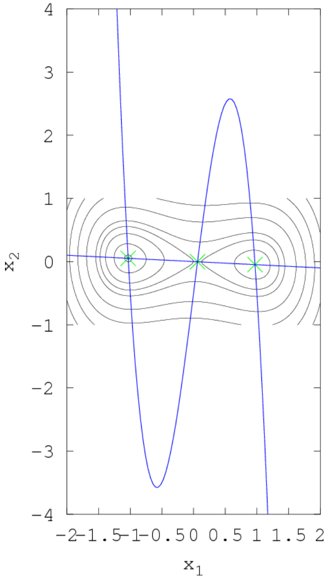
- "$\nabla f(x) = 0$" is a system of two equations in two unknowns:

$$\begin{aligned} 8x_1^3 - 8x_1 + \tfrac{1}{2} + x_2 &= 0, \\ x_1 + 20x_2 &= 0 \end{aligned}$$

- each of these equations is a (blue) curve in the x_1, x_2 plane



visualized equations $\nabla f(x) = 0$ ... more clearly



the short code that computes f, ∇f, and $\nabla^2 f$

pits.m

```
function [f, df, Hf] = pits(x)
% PITS Function with two local minima and one saddle.  Unique global minimum.

if length(x) ~= 2,  error('x must be length 2 vector'),  end
f = 2.0 * x(1)^4 - 4.0 * x(1)^2 + 10.0 * x(2)^2 + 0.5 * x(1) + x(1) * x(2);
df = [8.0 * x(1)^3 - 8.0 * x(1) + 0.5 + x(2);
      20.0 * x(2) + x(1)];
Hf = [24.0 * x(1)^2 - 8.0,  1.0;
      1.0,                  20.0];
end
```

- for use with optimization procedures it is best to have one code generate f and its derivatives
- all of these are allowed in MATLAB:

```
>> f = pits(x)
>> [f, df] = pits(x)
>> [f, df, Hf] = pits(x)
```

backtracking code `bt.m` (posted online)

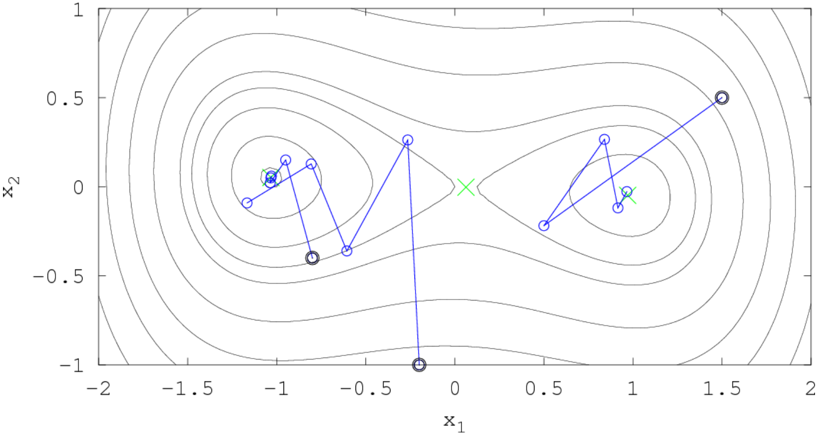
```
function alphak = bt(xk,pk,f,dfxk, ...
                     alphabar,c,rho)
% BT  Use backtracking to compute fractional step length alphak.
% ...
Dk = dfxk' * pk;
if Dk >= 0.0
    error('pk is not a descent direction ... stopping')
end

% set defaults according to which inputs are missing
if nargin < 6,  alphabar = 1.0;  end
if nargin < 7,  c = 1.0e-4;  end
if nargin < 8,  rho = 0.5;  end

% implement Algorithm 3.1
alphak = alphabar;
while f(xk + alphak * pk) > f(xk) + c * alphak * Dk
    alphak = rho * alphak;
end
```

- note how it sets defaults

steepest descent + back-tracking

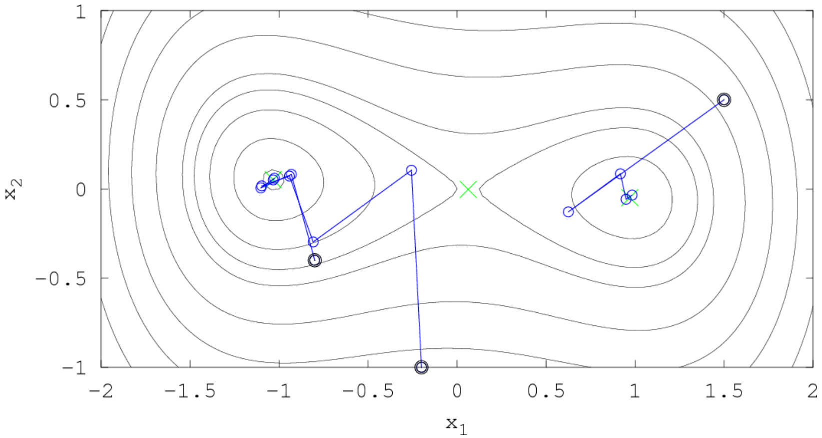


- choose three starting points
  $x_0 = (1.5, 0.5), (-0.2, -1), (-0.8, -0.4)$
- use steepest descent search vector:

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

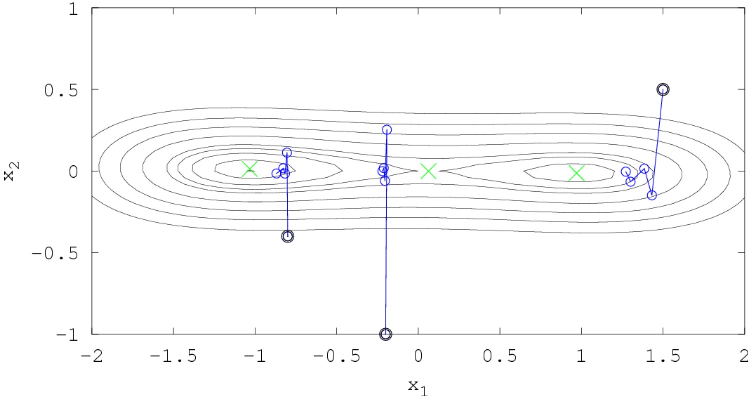
- works pretty well once contours are round

steepest descent + back-tracking: sensitive to scaling



- suppose we scale output of f: $\tilde{f}(x) = 7f(x)$
- changes behavior (in this case for the better . . .)

steepest descent + back-tracking: sensitive to scaling, cont.

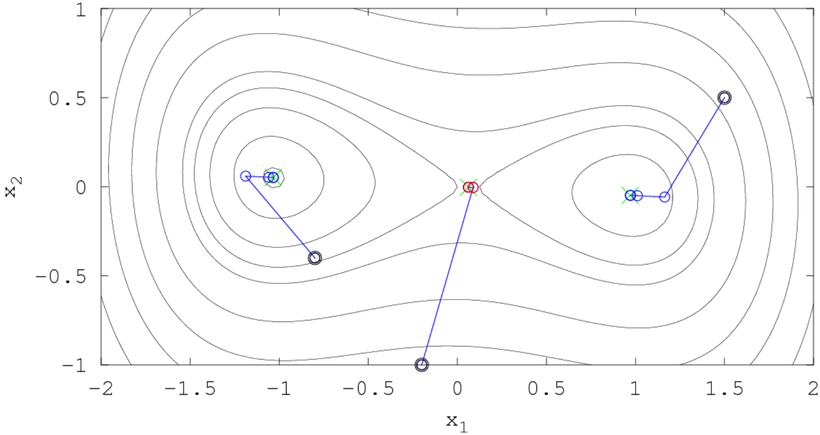


- this time, optimize the "same function" but with x_2 scaled:

$$\hat{f}(x) = f\left(\begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)$$

- not so good: non-round contours $\implies$ gradient not right direction
- *important idea*: steepest descent result affected by scaling of either x or $f(x)$

Newton + back-tracking

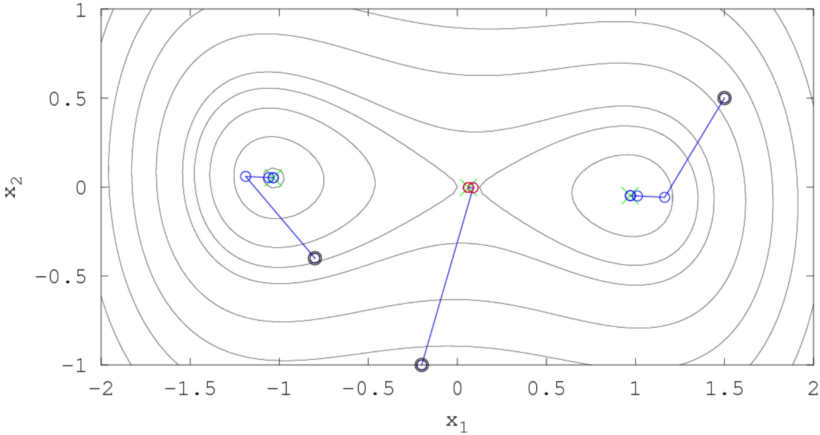


- redo last three slides but with Newton step:

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

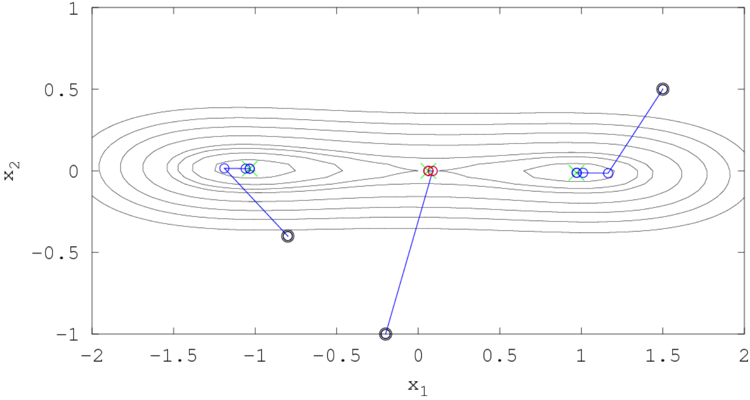
- red ○ are x_k where p_k is not descent direction

Newton + back-tracking: scale invariant



- now scale output of f: $\tilde{f}(x) = 7f(x)$

Newton + back-tracking: scale invariant



- now scale x_2:

$$\hat{f}(x) = f\left(\begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)$$

- *important idea*: Newton is invariant with changes with scaling of either x or $f(x)$