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

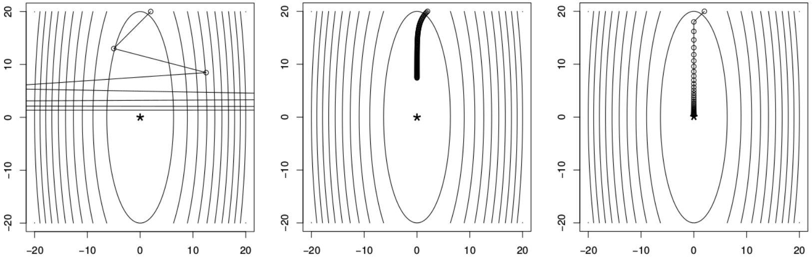
Ed Bueler

Math 661 Optimization

September 27, 2016

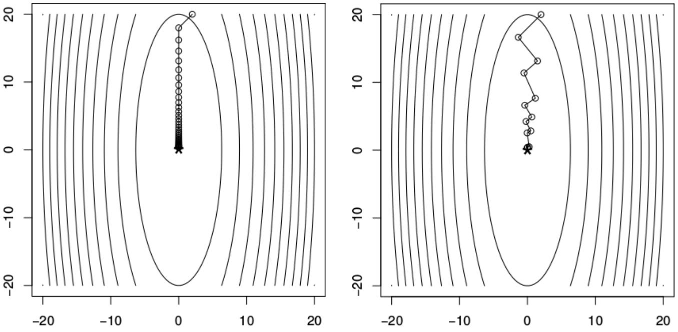
steepest descent with fixed steps

- 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 $\alpha_k = \tilde{\alpha}$: can get overshoot (*left*) or many steps (*middle*)
- ...one might "hand-tune" $\tilde{\alpha}$ for reasonable result (*right*)



steepest descent: back-tracking seems to help

- “hand-tuned” (*left*) and back-tracking (*right*) results seem to be comparable in number of steps
  - back-tracking shown in a few slides
- *main question*: is steepest-descent + back-tracking a good algorithm?
  - . . . remember that for *this* function, which is quadratic $f(x) = \frac{1}{2}x^\top Qx$, the Newton method converges in one step

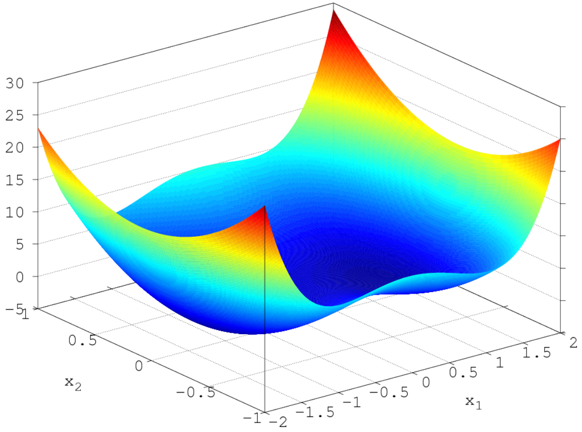


a more interesting example function

- consider this function:

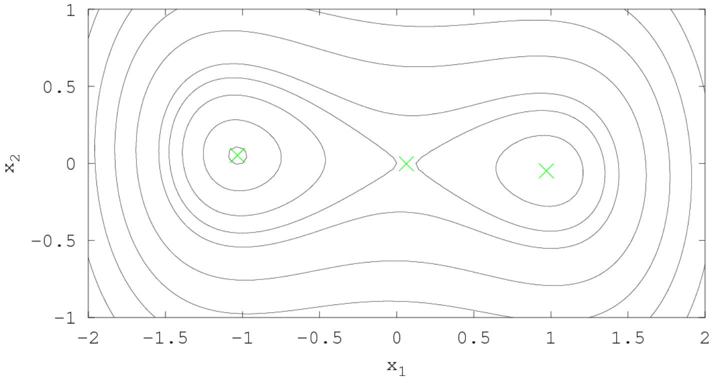
$$f(x) = 2x_1^4 - 4x_1^2 + 10x_2^2 + \frac{1}{2}x_1 + x_1x_2$$

  - quartic, but not "difficult" 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 $\times$)

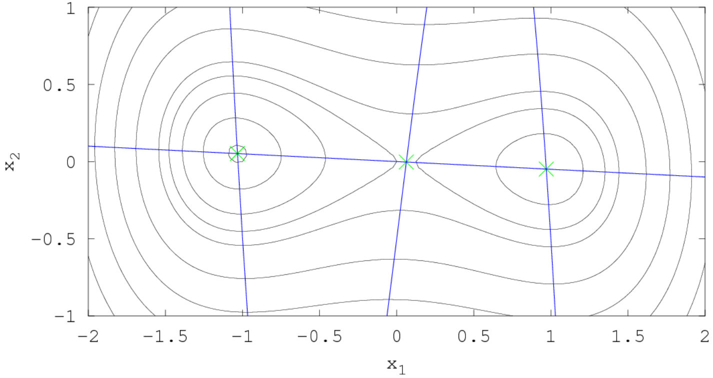


visualize equations $\nabla f(x) = 0$

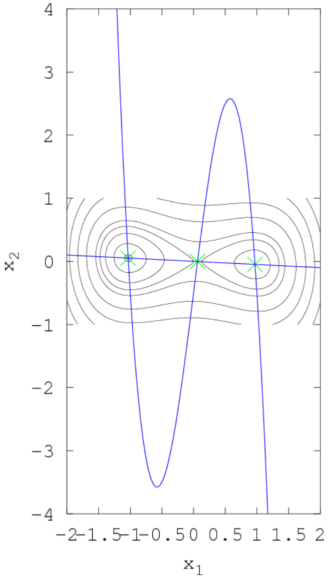
- "$\nabla f(x) = 0$" is a system of two equations in two unknowns:

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

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



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



code `pits.m` (posted online) 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 most 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)

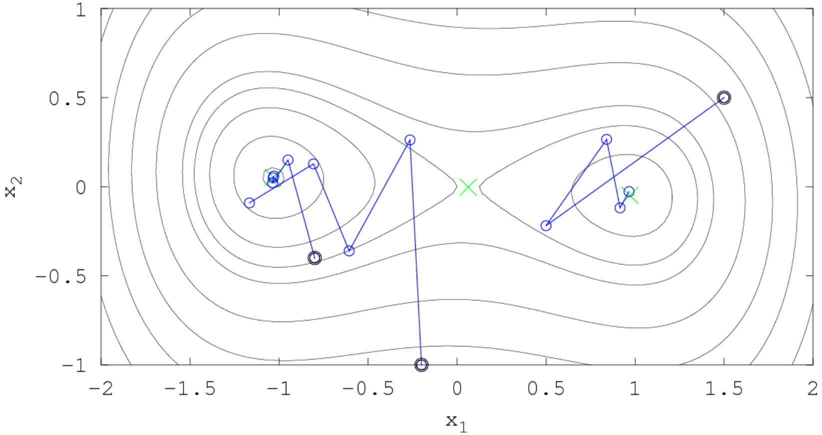
```
function alphak = bt(xk,pk,f,dfxk, ...
                     alphabar,c,rho)
% BT  Use backtracking to compute fractional step length alphak.
% ...
Dk = dfxk' * pk;
if Dk >= 0.0
    error('pk is not a descent direction ... stopping')
end

% 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

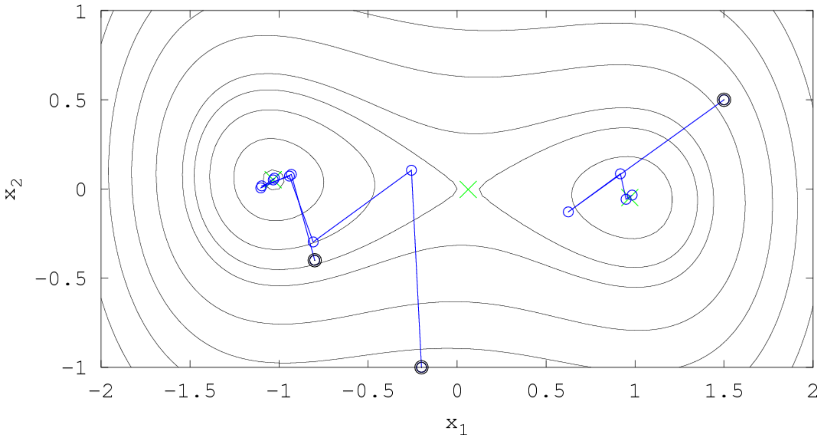


- choose three starting points
  $x_0 = (1.5, 0.5), (-0.2, -1), (-0.8, -0.4)$
- use steepest descent search vector:

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

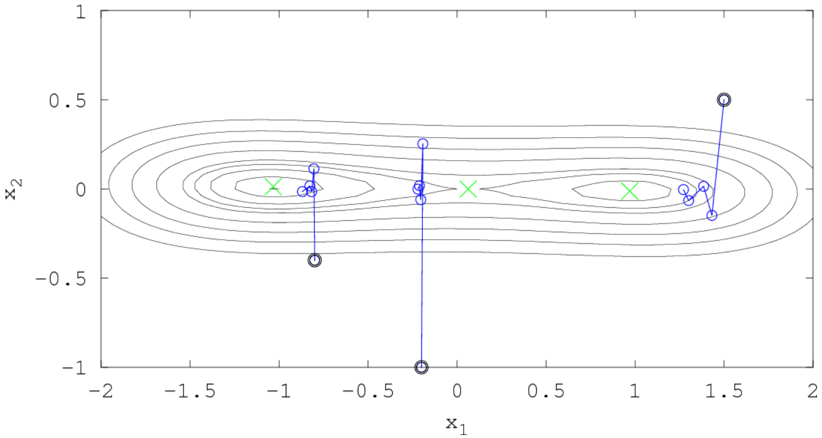
- works pretty well once contours are round

steepest descent + back-tracking: sensitive to scaling



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

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

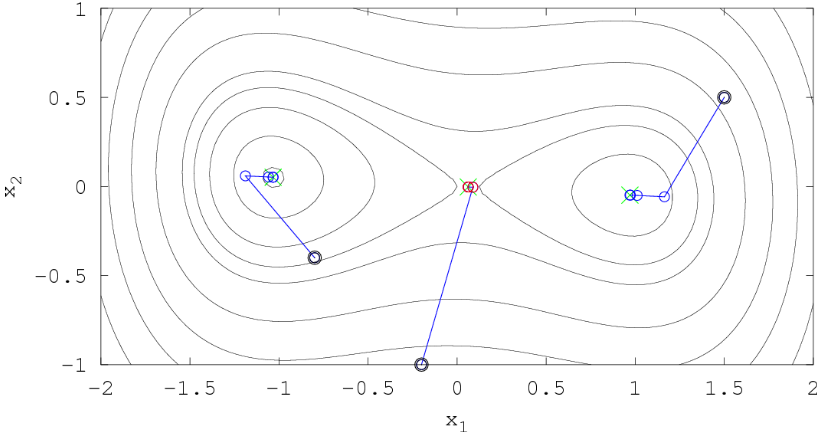


- this time, optimize the “same function” but with input x_2 scaled:

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

- not so good: non-round contours $\implies$ gradient not right direction

Newton + back-tracking

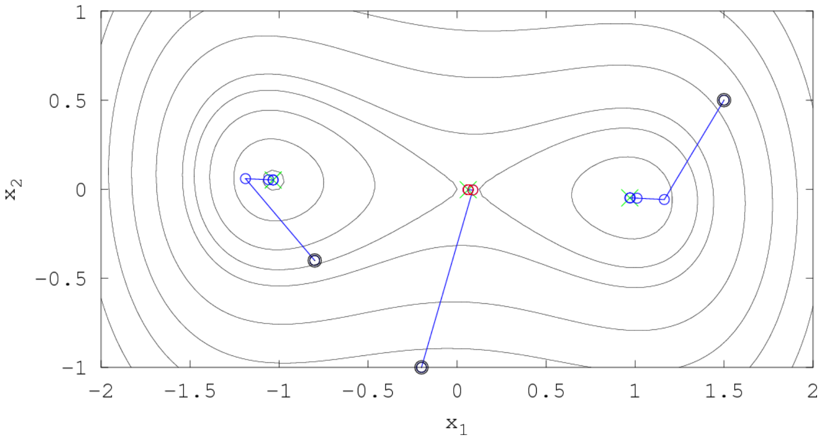


- redo last three slides but with Newton step:

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

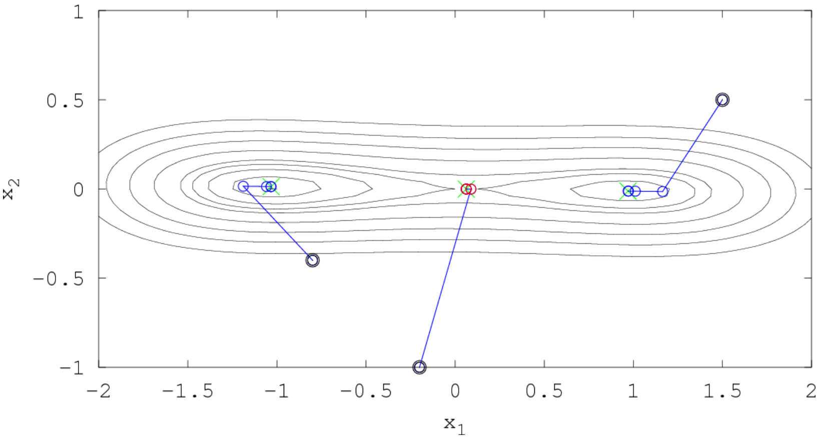
- red ∘ are x_k where p_k is not a descent direction

Newton + back-tracking: output-scale invariant



- now scale output of f: $\hat{f}(x) = 7f(x)$
- makes *no* difference; why?

Newton + back-tracking: input-scale insensitive



- now scale input x_2:

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

- makes *no* difference; why?

conclusions: steepest descent

- steepest descent results are significantly affected by scaling of either x or $f(x)$
- back-tracking helps with performance but does not address (or fix) the scaling sensitivity

conclusions: Newton

- Newton is invariant to scaling of output $f(x)$: if $\hat{f}(x) = \lambda f(x)$ and $\lambda > 0$ then

$$\begin{aligned}\hat{p}_k &= -\nabla^2 \hat{f}(x_k)^{-1} \nabla \hat{f}(x_k) = -\left(\lambda \nabla^2 f(x_k)\right)^{-1} (\lambda \nabla f(x_k)) \\ &= -\nabla^2 f(x_k)^{-1} \nabla f(x_k) = p_k\end{aligned}$$

- Newton is invariant to scaling of input x: if $\tilde{f}(z) = f(Sz)$ and $S \in \mathbb{R}^{n \times n}$ is invertible then

$$\begin{aligned}\tilde{p}_k &= -\nabla^2 \tilde{f}(z_k)^{-1} \nabla \tilde{f}(z_k) = -\left(S^\top \nabla^2 f(Sx_k) S\right)^{-1} (S^\top \nabla f(Sx_k)) \\ &= -S^{-1} \nabla^2 f(Sx_k)^{-1} (S^\top)^{-1} S^\top \nabla f(Sx_k) \\ &= -S^{-1} \nabla^2 f(Sx_k)^{-1} \nabla f(Sx_k) = S^{-1} p_k,\end{aligned}$$

  by Exercise 2.10, so

$$x_{k+1} = Sz_{k+1} = Sz_k + S\tilde{p}_k = x_k + SS^{-1} p_k = x_k + p_k$$