Optimization 101

CSE P576

David M. Rosen

Recap

First half:

- Photogrammetry and bundle adjustment
- Maximum likelihood estimation

This half:

- Basic theory of optimization

  (i.e. how to *actually do* MLE)

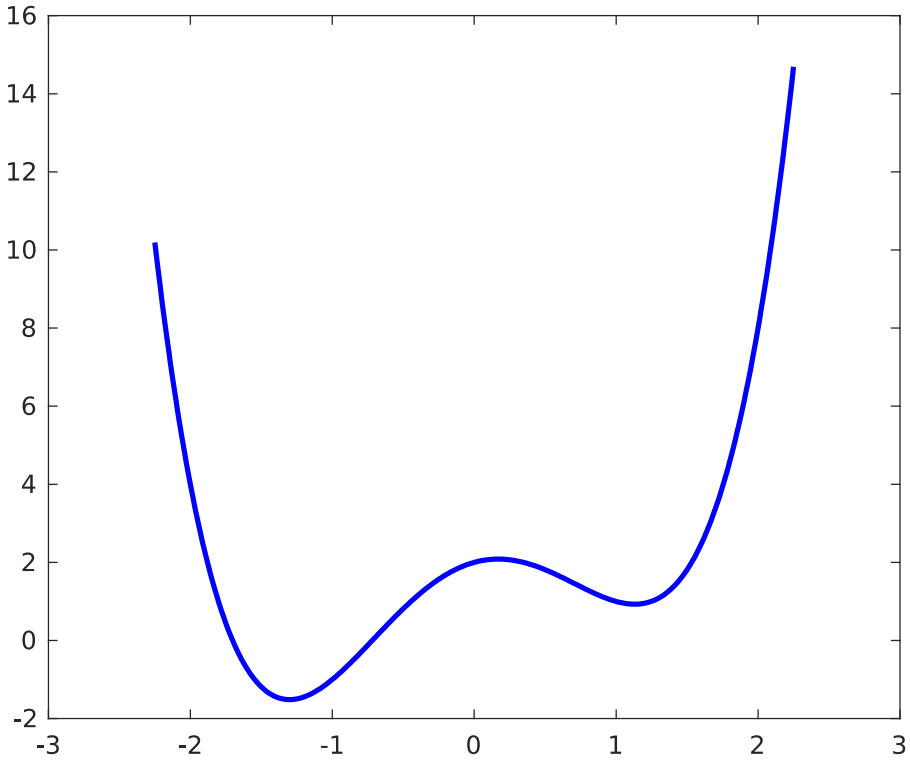
The Main Idea

Given: $f: R^n \rightarrow R$, we want to

$$\min_x f(x)$$

Problem: We have *no idea* how to actually do this …

Main idea: Let's *approximate f* with a simple *model function m*, and use that to search for a minimizer of *f*.



Optimization Meta-Algorithm

Given: A function $f: R^n \to R$ and an initial guess $x_0 \in R^n$ for a minimizer

Iterate:

- Construct a *model* $m_i(h) \approx f(x_i + h)$ of f near x_i.
- Use m_i to search for a *descent direction h* ($f(x + h) < f(x)$)
- Update $x_{i+1} \leftarrow x_i + h$

until convergence

A first example

Let's consider applying the Basic Algorithm to minimize

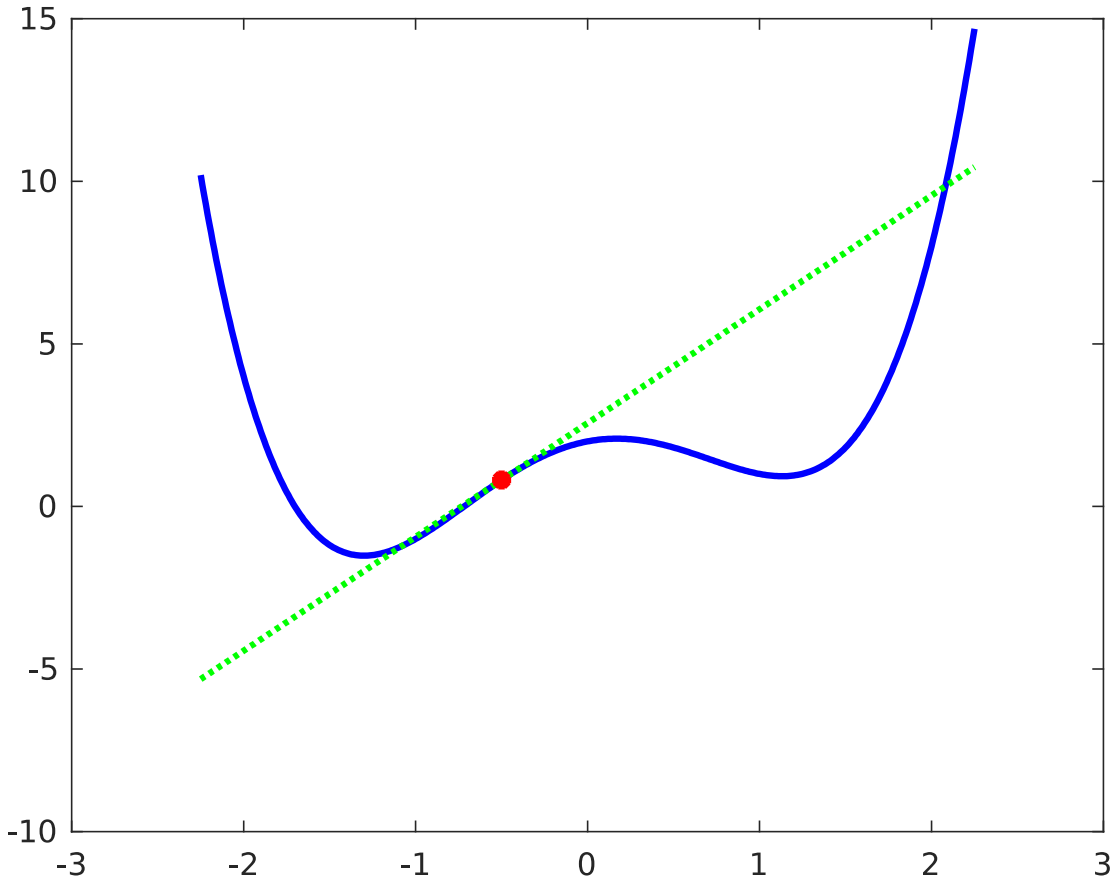
$$f(x) = x^4 - 3x^2 + x + 2$$

starting at $x_0 = -\frac{1}{2}$.

Q: How can we approximate (*model*) f near x_0?

A: Let's try linearizing! Take

$$m_0(h) \triangleq f(x_0) + f'(x_0)h$$



Gradient descent

Given:

- A function $f: R^n \to R$
- An initial guess $x_0 \in R^n$ for a minimizer
- Sufficient decrease parameter $c \in (0,1)$, stepsize shrinkage parameter $\tau \in (0,1)$
- Gradient tolerance $\epsilon > 0$

Iterate:

- Compute search direction $p = -\nabla f(x_i)$ at x_i
- Set initial stepsize $\alpha = 1$
- *Backtracking line search*: Update $\alpha \leftarrow \tau\alpha$ until the *Armijo-Goldstein sufficient decrease condition*:

$$f(x_i + \alpha p) < f(x_i) - c\alpha\|p\|^2$$

  is satisfied
- Update $x_{i+1} \leftarrow x_i + \alpha p$

until $\|\nabla f(x_i)\| < \epsilon$

Exercise: Minimizing a quadratic

Try minimizing the quadratic:

$$f(x, y) = x^2 - xy + \kappa y^2$$

using gradient descent, starting at $x_0 = (1,1)$ and using $c, \tau = \frac{1}{2}$ and $\epsilon = 10^{-3}$, for a few different values of κ, say

$$\kappa \in \{1, 10, 100, 1000\}$$

Q: If you plot function value $f(x_i)$ vs. iteration number *i*, what do you notice?

Gradient Descent

Given:

- A function $f: R^n \to R$
- An initial guess $x_0 \in R^n$ for a minimizer
- Sufficient decrease parameter $c \in (0,1)$, stepsize shrinkage parameter $\tau \in (0,1)$
- Gradient tolerance $\epsilon > 0$

Iterate:

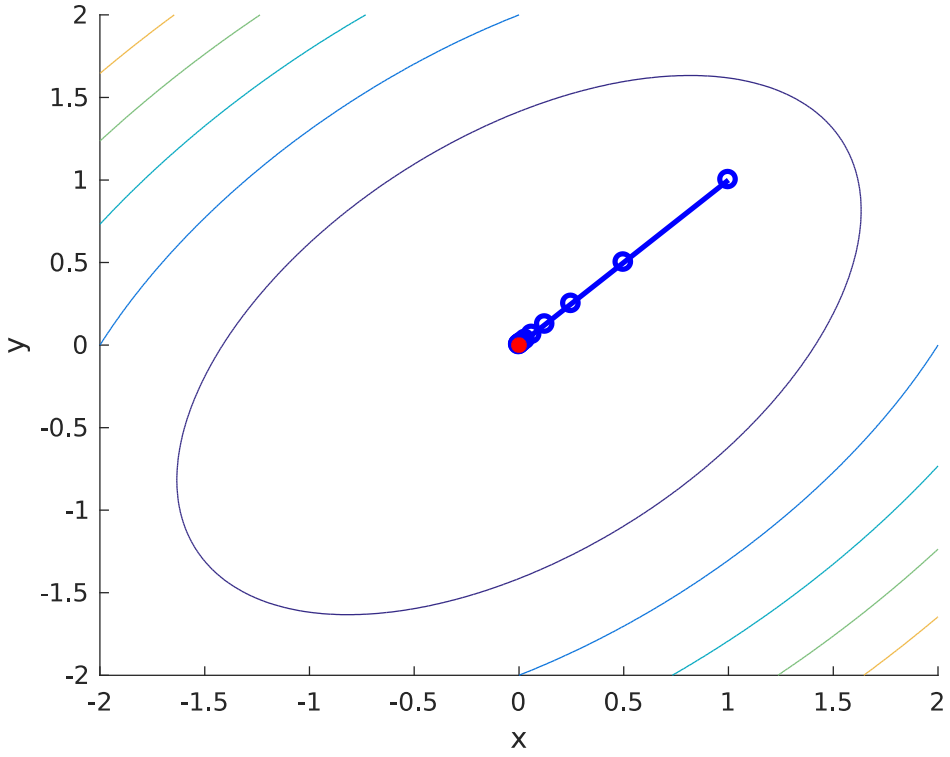
- Compute search direction $p = -\nabla f(x_i)$ at x_i
- Set initial stepsize $\alpha = 1$
- Line search: update $\alpha \leftarrow \tau\alpha$ until

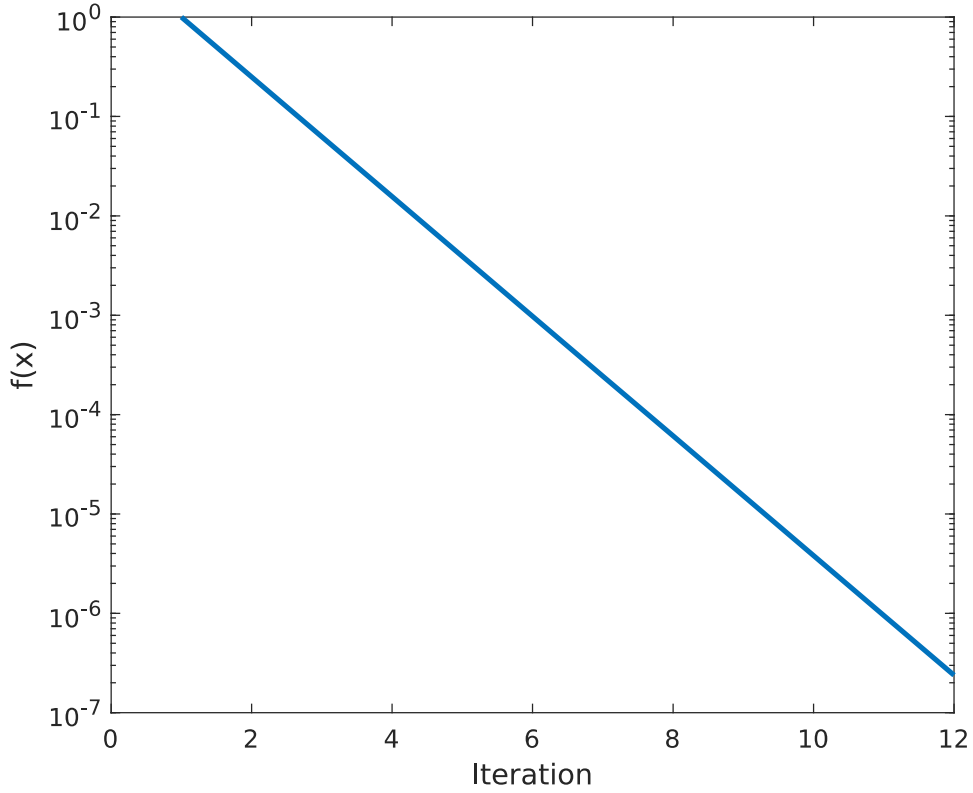
$$f(x_i + \alpha p) < f(x_i) - c\alpha\|p\|^2$$

- Update $x_{i+1} \leftarrow x_i + \alpha p$

until $\|\nabla f(x_i)\| < \epsilon$

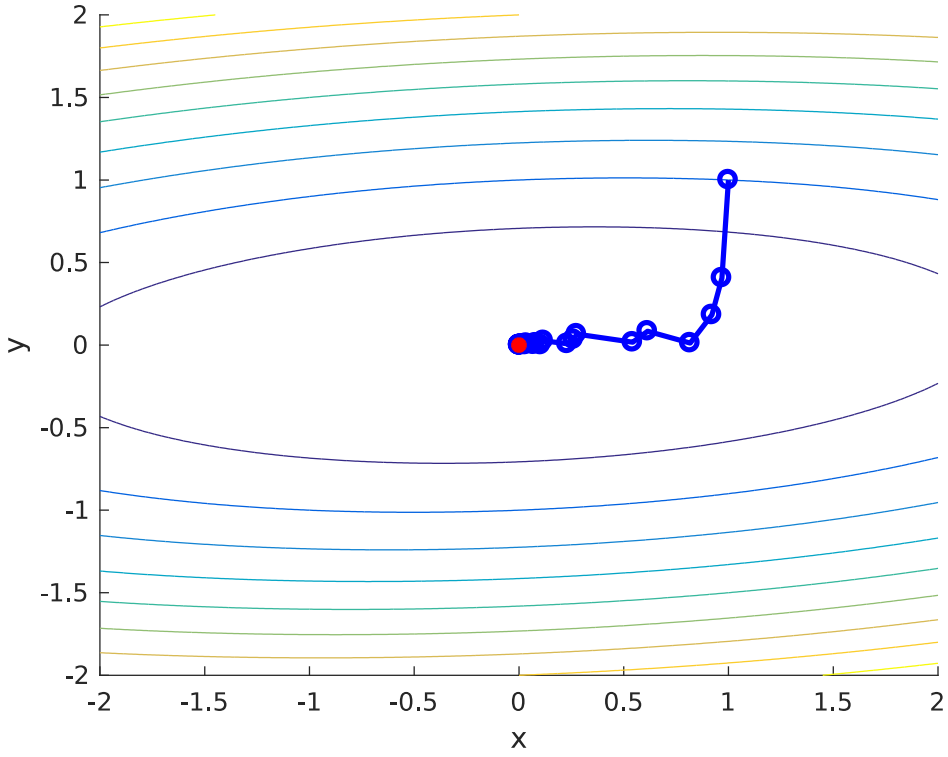
Exercise: Minimizing a quadratic

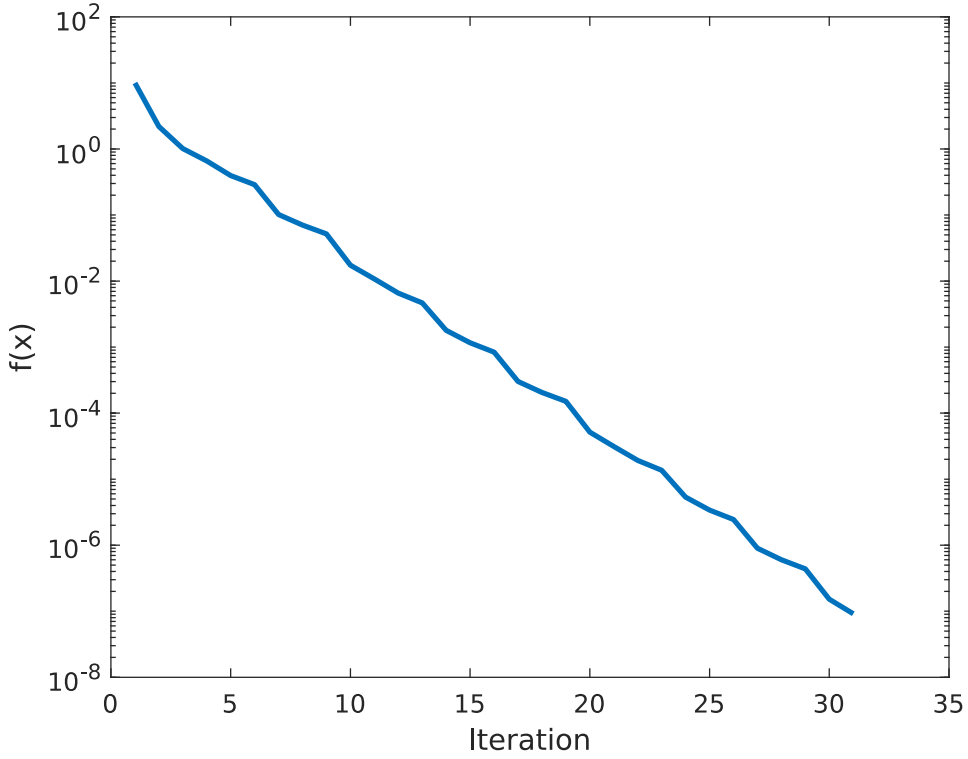




$\kappa = 1$

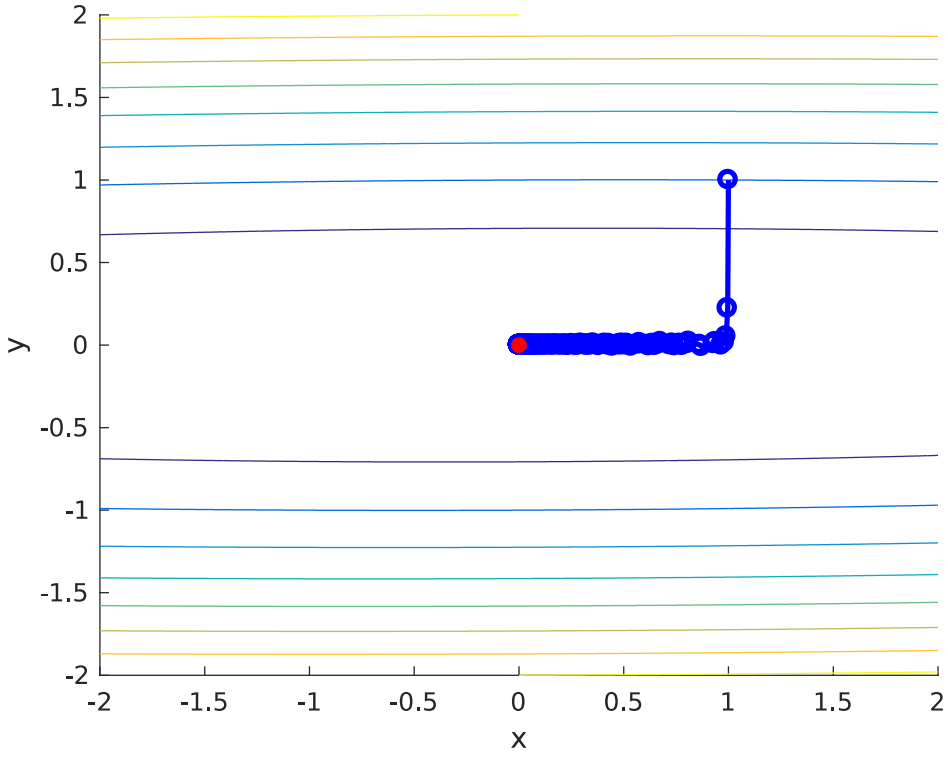
Exercise: Minimizing a quadratic

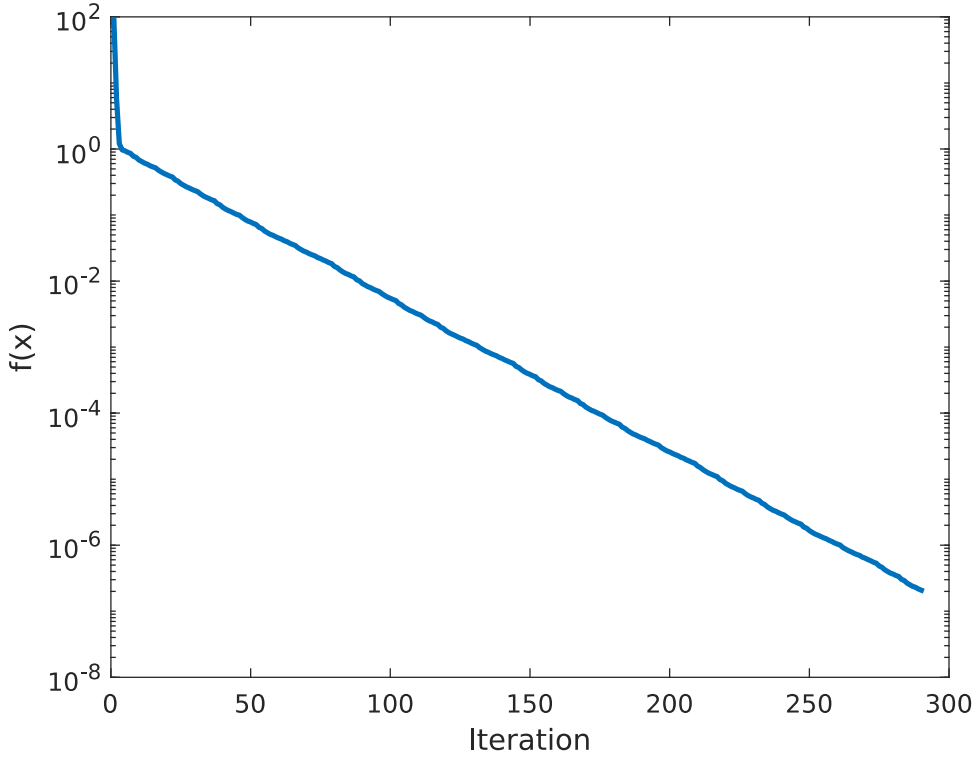




$\kappa = 10$

Exercise: Minimizing a quadratic





$\kappa = 100$

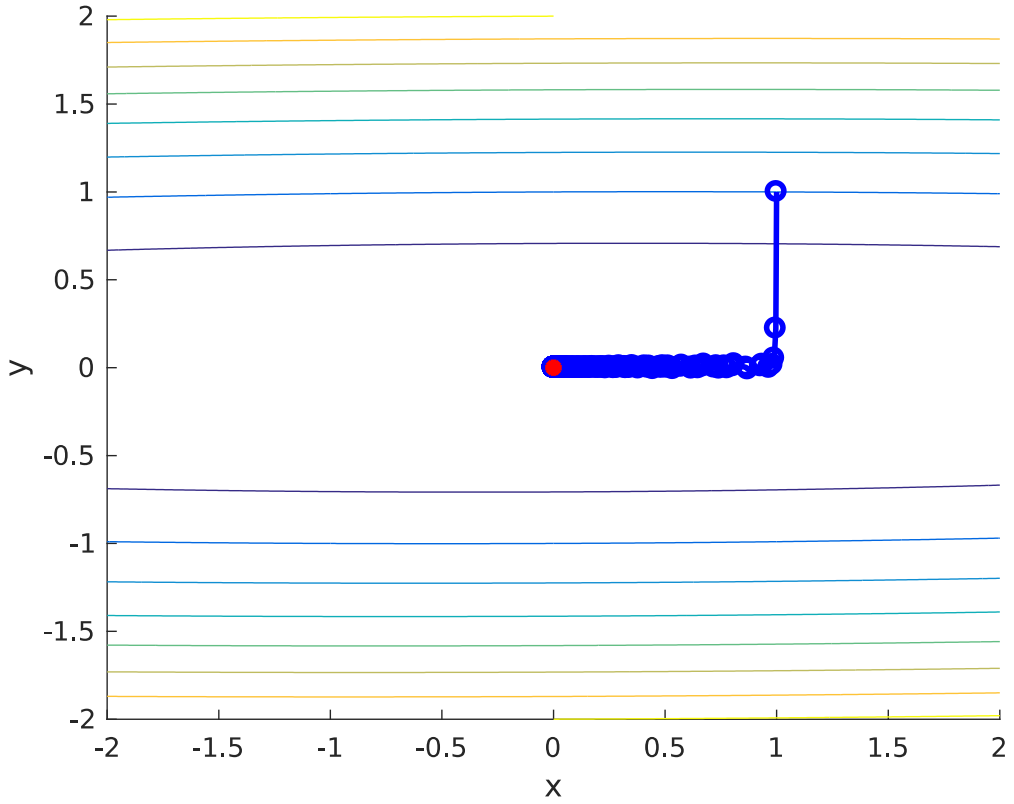
The problem of conditioning

Gradient descent doesn't perform well when f is *poorly conditioned* (has "stretched" contours).

Q: How can we improve our local model:

$$m_i(h) = f(x_i) + \nabla f(x_i)^T h$$

so that it handles curvature better?




WE NEED TO GO
DEEPER

Second-order methods

Let's try adding in curvature information using a *second-order* model for *f*:

$$m_i(h) = f(x_i) + \nabla f(x_i)^T h + \frac{1}{2} h^T \nabla^2 f(x) h$$

NB: If $\nabla^2 f(x) \succ 0$, then $m_i(h)$ has a *unique minimizer*:

$$h_N = -\left(\nabla^2 f(x_0)\right)^{-1} \nabla f(x_0)$$

In that case, using the update rule:

$$x_{i+1} \leftarrow x_i + h_N$$

gives ***Newton's method***

Exercise: Minimizing a quadratic

Let's try minimizing the quadratic:

$$f(x, y) = x^2 - xy + \kappa y^2$$

again, this time using Newton's method, starting at $x_0 = (1,1)$ and using $\epsilon = 10^{-3}$, for

$$\kappa \in \{1, 10, 100, 1000\}$$

If you plot function value $f(x_i)$ vs. iteration number *i*, what do you notice?

Newton's method

Given:

- A function $f: R^n \to R$
- An initial guess $x_0 \in R^n$ for a minimizer
- Gradient tolerance $\epsilon > 0$

Iterate:

- Compute gradient $\nabla f(x_i)$ and Hessian $\nabla^2 f(x_i)$
- Compute Newton step:
$$h_N = -\left(\nabla^2 f(x_0)\right)^{-1} \nabla f(x_0)$$
- Update $x_{i+1} \leftarrow x_i + h_N$
- **until** $\|\nabla f(x_i)\| < \epsilon$

Quasi-Newton methods

Newton's method is ***fast***! (It has a *quadratic* convergence rate)

But:

- h_N is only guaranteed to be a descent direction if $\nabla^2 f(x_i) \succ 0$
- Computing exact Hessians can be expensive!

Quasi-Newton methods: Use a *positive-definite approximate Hessian* B_i in the model function:

$$m_i(h) = f(x_i) + \nabla f(x_i)^T h + \frac{1}{2} h^T B_i h$$

$\Rightarrow m_i(h)$ *always* has a unique minimizer:

$$h_{QN} = -B_i^{-1} \nabla f(x_i)$$

$\Rightarrow h_{QN}$ is *always* a descent direction!

Quasi-Newton method with line search

Given:

- A function $f: R^n \to R$
- An initial guess $x_0 \in R^n$ for a minimizer
- Sufficient decrease parameter $c \in (0,1)$, stepsize shrinkage parameter $\tau \in (0,1)$
- Gradient tolerance $\epsilon > 0$

Iterate:

- Compute gradient $g_i = \nabla f(x_i)$ and positive-definite Hessian approximation B_i at x_i
- Compute quasi-Newton step:

$$h_{QN} = -B_i^{-1} g_i$$

- Set initial stepsize $\alpha = 1$
- *Backtracking line search*: Update $\alpha \leftarrow \tau\alpha$ until the *Armijo-Goldstein sufficient decrease condition*:

$$f(x_i + \alpha h_{QN}) < f(x_i) + c\alpha g_i^T h_{QN}$$

  is satisfied
- Update $x_{i+1} \leftarrow x_i + \alpha h_{QN}$

until $\|g_i\| < \epsilon$

Quasi-Newton methods (cont'd)

Different choices of B_i give different QN algorithms

$\Rightarrow$ Can trade off *accuracy* of B_i with *computational cost*

LOTS of possibilities here!

- Gauss-Newton
- Levenberg-Marquardt
- (L-) BFGS
- Broyden
- etc …

$\Rightarrow$ Don't be afraid to experiment ☺!

Special case: The Gauss-Newton method

A quasi-Newton algorithm for minimizing a *nonlinear least-squares objective*:

$$f(x) = \|r(x)\|^2$$

Uses the local quadratic model obtained by *linearizing r*:

$$m_i(h) = \|r(x_i) + J(x_i)h\|^2$$

where $J(x_i) \triangleq \frac{\partial r}{\partial x}(x_i)$ is the *Jacobian* of *r*.

Equivalently:

$$g_i = 2J(x_i)^T r(x_i), \qquad B_i = 2J(x_i)^T J(x_i)$$

A word on linear algebra

The dominant cost (memory + time) in a QN method is *linear algebra*:

- Constructing the Hessian approximation B_i
- Solving the linear system:

$$B_i h_{QN} = -h_{QN}$$

$\Rightarrow$ Fast/robust linear algebra is ***essential*** for efficient QN methods

- Take advantage of sparsity in B_i!

- **NEVER, NEVER, NEVER INVERT B_i directly!!!**
  - It's incredibly expensive and unnecessary
  - **Use instead** [cf. Golub & Van Loan's *Matrix Computations*]:
    - *Matrix factorizations:* QR, Cholesky, LDL^T
    - *Iterative linear methods:* conjugate gradient

A word on linear algebra

NEVER INVERT B_i!!!

Optimization methods: Cheat sheet

First-order methods

Use only gradient information

- **Pro:** Local model is inexpensive
- **Con:** Slow (linear) convergence rate

Canonical example: Gradient descent

Best for:

- Moderate accuracy
- Very large problems

Second-order methods

Use (some) 2nd-order information

- **Pro:** Fast (superlinear) convergence
- **Con:** Local model can be expensive

Canonical example: Newton's method

Best for:

- High accuracy
- Small to moderately large problems