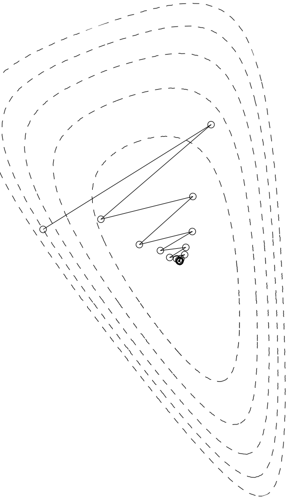


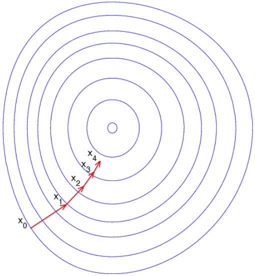
Optimization Algorithms

Unconstraint Optimization Basics

Descent direction & stepsize, plain gradient descent, stepsize adaptation & backtracking line search, trust region, steepest descent, Newton, Gauss-Newton, Quasi-Newton, BFGS, conjugate gradient, exotic: Rprop

Marc Toussaint
Technical University of Berlin
Winter 2020/21

Gradient descent



- Objective function: $f: \mathbb{R}^n \to \mathbb{R}$
  Gradient vector: $\nabla f(x) = \left[\frac{\partial}{\partial_x} f(x)\right]^\top \in \mathbb{R}^n$

- Problem:

$$\min_x f(x)$$

  where we can evaluate $f(x)$ and $\nabla f(x)$ for any $x \in \mathbb{R}^n$

- Plain gradient descent: iterative steps in the direction $-\nabla f(x)$.

Input: initial $x \in \mathbb{R}^n$, function $\nabla f(x)$, stepsize α, tolerance θ
Output: x

repeat
    $x \leftarrow x - \alpha \nabla f(x)$
until $|\Delta x| < \theta$ [perhaps for 10 iterations in sequence]

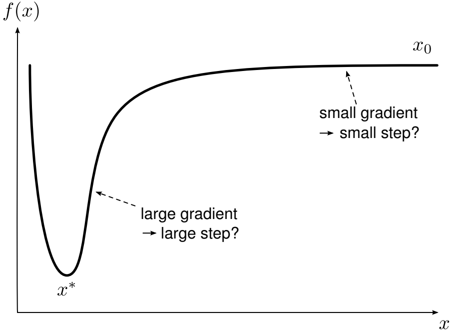
- Plain gradient descent is really not efficient
- Two core issues of unconstrainted optimization:

A. Stepsize

B. Descent direction

Stepsize

- Making steps proportional to $\nabla f(x)$?



- We need methods that
  – robustly adapt stepsize
  – exploit convexity, if known
  – perhaps be independent of $|\nabla f(x)|$ (e.g. if non-convex as above)

Stepsize Adaptation: Backtracking Line Search

Input: initial $x \in \mathbb{R}^n$, functions $f(x)$ and $\nabla f(x)$, tolerance θ, parameters (defaults: $\varrho^+_\alpha = 1.2, \varrho^-_\alpha = 0.5, \delta_{\text{max}} = \infty, \varrho_{\text{ls}} = 0.01$)

initialize stepsize $\alpha = 1$
repeat
    $\delta \leftarrow -\frac{\nabla f(x)}{|\nabla f(x)|}$ *// (alternative: $\delta = -\nabla f(x)$)*
    while $f(x + \alpha\delta) > f(x) + \varrho_{\text{ls}} \nabla f(x)^\top (\alpha\delta)$ **do** ***// line search***
        $\alpha \leftarrow \varrho^-_\alpha \alpha$ *// decrease stepsize*
    end while
    $x \leftarrow x + \alpha\delta$
    $\alpha \leftarrow \min\{\varrho^+_\alpha \alpha, \delta_{\text{max}}\}$ *// increase stepsize*
until $|\alpha\delta| < \theta$ *// perhaps for 10 iterations in sequence*

- α determines the absolute stepsize
- Guaranteed monotonicity (by construction)
  ("Typically" ensures convergence to locally convex minima; see later)

Backtracking line search

- Line search in general denotes the problem

$$\min_{\alpha \geq 0} f(x + \alpha\delta)$$

  for some step direction δ.

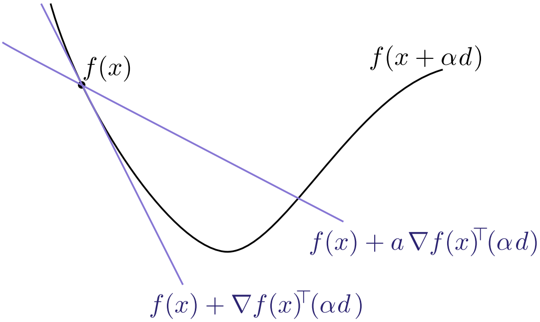
- The most common line search is **backtracking**, which decreases α as long as

$$f(x + \alpha\delta) > f(x) + \varrho_{\textsf{ls}} \nabla f(x)^\top (\alpha\delta)$$

  ϱ_α^- describes the stepsize decrement in case of a rejected step
  $\varrho_{\textsf{ls}}$ describes a minimum desired decrease in $f(x)$

- Boyd at al: typically $\varrho_{\textsf{ls}} \in [0.01, 0.3]$ and $\varrho_\alpha^- \in [0.1, 0.8]$

Backtracking line search



Wolfe Conditions

- The 1st Wolfe condition ("sufficient decrease condition")

$$f(x + \alpha\delta) \le f(x) + \varrho_{\text{ls}} \nabla f(x)^\top (\alpha\delta)$$

requires a decrease of f at least ϱ_{ls}-times "as expected"

- The 2nd (stronger) Wolfe condition ("curvature condition")

$$|\nabla f(x + \alpha\delta)^\top \delta| \le \varrho_{\text{ls2}} |\nabla f(x)^\top \delta|$$

implies a requires an decrease of the slope by a factor ϱ_{ls2}.
$\varrho_{\text{ls2}} \in (\varrho_{\text{ls}}, \frac{1}{2})$ (for conjugate gradient)

- See Nocedal et al., Section 3.1 & 3.2 for more general proofs of convergence of any method that ensures the Wolfe conditions after each line search

Convergence for (locally) convex functions

- **Theorem** (Exponential convergence on convex functions)
  - Let $f : \mathbb{R}^n \to \mathbb{R}$ be an objective function
  - with eigenvalues λ of the Hessian $\nabla^2 f(x)$ bounded by $m < \lambda < M$, with $m > 0$, $\forall x \in \mathbb{R}^n$
  - then gradient descent with backtracking line search converges exponentially with convergence rate $(1 - 2\frac{m}{M}\varrho_{\text{ls}}\varrho_\alpha^-)$.

- The exercises guide through the proof

Newton Methods & Descent Direction

Steepest Descent Direction

- The gradient $-\nabla f(x)$ is sometimes called *steepest descent direction*

Is it really?

Steepest Descent Direction

- The gradient $-\nabla f(x)$ is sometimes called *steepest descent direction*

Is it really?

- Here is a possible definition:

The steepest descent direction is the one where, when you make a step of length 1, you get the largest decrease of f in its linear approximation.

$$\underset{\delta}{\operatorname{argmin}}\, \nabla f(x)^\top \delta \qquad \text{s.t. } \|\delta\| = 1$$

Steepest Descent Direction

- But the norm $\|\delta\|^2 = \delta^\top A \delta$ depends on the metric A!

  Let $A = B^\top B$ (Cholesky decomposition) and $z = B\delta$

$$\begin{aligned}
\delta^* &= \underset{\delta}{\operatorname{argmin}}\, \nabla f^\top \delta \qquad \text{s.t. } \delta^\top A \delta = 1 \\
&= B^{\text{-}1} \underset{z}{\operatorname{argmin}} (B^{\text{-}1} z)^\top \nabla f \qquad \text{s.t. } z^\top z = 1 \\
&= B^{\text{-}1} \underset{z}{\operatorname{argmin}}\, z^\top B^{\text{-}\top} \nabla f \qquad \text{s.t. } z^\top z = 1 \\
&= B^{\text{-}1} [-B^{\text{-}\top} \nabla f] = -A^{\text{-}1} \nabla f
\end{aligned}$$

- The steepest descent direction is $\delta = -A^{\text{-}1} \nabla f$

Behavior under linear coordinate transformations

- Let B be a matrix that describes a linear transformation in coordinates

- A coordinate vector x transforms as $z = Bx$
- The gradient vector $\nabla_x f(x)$ transforms as $\nabla_z f(z) = B^{-\top} \nabla_x f(x)$
- The metric A transforms as $A_z = B^{-\top} A_x B^{-1}$
- The steepest descent transforms as $A_z^{-1} \nabla_z f(z) = B A_x^{-1} \nabla_x f(x)$

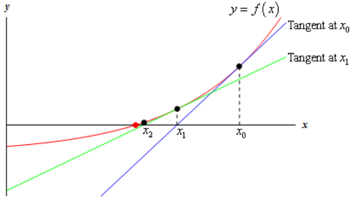
  The steepest descent transforms like a normal coordinate vector (covariant)

(more details in the *Maths script*)

Newton Step

Newton Step

- For finding roots (zero points) of $f(x)$



$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

- For finding optima of $f(x)$ in 1D (which are roots of $f'(x)$):

$$x \leftarrow x - \frac{f'(x)}{f''(x)}$$

- For finding optima in higher dimensions $x \in \mathbb{R}^n$:

$$x \leftarrow x - \nabla^2 f(x)^{-1} \nabla f(x)$$

Newton Step

- Assume we have access to the symmetric **Hessian**

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2}{\partial_{x_1}\partial_{x_1}} f(x) & \frac{\partial^2}{\partial_{x_1}\partial_{x_2}} f(x) & \cdots & \frac{\partial^2}{\partial_{x_1}\partial_{x_n}} f(x) \\ \frac{\partial^2}{\partial_{x_1}\partial_{x_2}} f(x) & & & \vdots \\ \vdots & & & \vdots \\ \frac{\partial^2}{\partial_{x_n}\partial_{x_1}} f(x) & \cdots & \cdots & \frac{\partial^2}{\partial_{x_n}\partial_{x_n}} f(x) \end{pmatrix} \in \mathbb{R}^{n\times n}$$

- which defines the Taylor expansion:

$$f(x+\delta) \approx f(x) + \nabla f(x)^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f(x)\ \delta$$

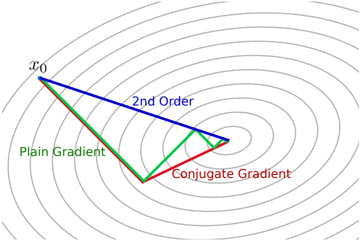
Note: $\nabla^2 f(x)$ acts like a **metric** for δ

Notes on the Newton Step

- If f is a 2nd-order polynomial, the Newton step jumps to the optimum in just one step.
- *Unlike the gradient magnitude* $|\nabla f(x)|$, the magnitude of the Newton step δ is meaningful and scale invariant!
  - If you'd rescale f (trade cents by Euros), δ is unchanged
  - $|\delta|$ is the distance to the optimum of the 2nd-order Taylor.
- *Unlike the gradient* $\nabla f(x)$, the Newton step δ is truely a vector!
  - The Newton step is invariant under coordinate transformations; the coordinates of δ transforms contra-variant, as it is supposed to for vector coordinates
  - The proof is exactly the same as for the steepest descent with a non-Euclidean metric – the Hessian acts as a metric

Why 2nd order information is better

- Better direction:



- Better stepsize:
  - A full Newton step jumps directly to the minimum of the local squared approx.
  - Robust Newton methods combine this with line search and damping (Levenberg-Marquardt)

Basic Newton method

Input: initial $x \in \mathbb{R}^n$, functions $f(x), \nabla f(x), \nabla^2 f(x)$, tolerance θ, parameters (defaults: $\varrho_\alpha^+ = 1.2, \varrho_\alpha^- = 0.5, \varrho_{\text{ls}} = 0.01, \lambda$)

1: initialize stepsize $\alpha = 1$, fixed damping λ
2: **repeat**
3: compute δ to solve $(\nabla^2 f(x) + \lambda \mathbf{I})\ \delta = -\nabla f(x)$
4: **while** $f(x + \alpha\delta) > f(x) + \varrho_{\text{ls}} \nabla f(x)^\top (\alpha\delta)$ **do** *// line search*
5: $\alpha \leftarrow \varrho_\alpha^- \alpha$ *// decrease stepsize*
6: **end while**
7: $x \leftarrow x + \alpha\delta$ *// step is accepted*
8: $\alpha \leftarrow \min\{\varrho_\alpha^+ \alpha, 1\}$ *// increase stepsize*
9: **until** $\|\alpha\delta\|_\infty < \theta$

- Notes:
  - Line 3 computes the Newton step $\delta = -\nabla^2 f(x)^{-1} \nabla f(x)$,
    e.g. using a special Lapack routine `dposv` to solve $Ax = b$ (using Cholesky)

Damping, λ, Trust-Region, Levenberg-Marquardt

- δ solves the problem

$$\min_{\delta}\left[\nabla f(x)^\top\delta + \frac{1}{2}\delta^\top\nabla^2 f(x)\delta + \frac{1}{2}\lambda\delta^2)\right] .$$

  - λ introduces a squared penalty for large steps

Damping, λ, Trust-Region, Levenberg-Marquardt

- δ solves the problem

$$\min_\delta \left[\nabla f(x)^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f(x) \delta + \frac{1}{2}\lambda \delta^2)\right] .$$

  - λ introduces a squared penalty for large steps

- **Trust region method:**

$$\min_\delta \left[\nabla f(x)^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f(x) \delta\right] \quad \text{s.t.} \quad \delta^2 \le \beta$$

  - β defines the *trust region*

Damping, λ, Trust-Region, Levenberg-Marquardt

- δ solves the problem

$$\min_\delta \left[\nabla f(x)^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f(x) \delta + \frac{1}{2}\lambda \delta^2)\right] .$$

  – λ introduces a squared penalty for large steps

- **Trust region method:**

$$\min_\delta \left[\nabla f(x)^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f(x) \delta\right] \quad \text{s.t.} \quad \delta^2 \le \beta$$

  – β defines the *trust region*

- Solving this using Lagrange parameters (as we will learn it later):

$$L(\delta, \lambda) = \nabla f(x)^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f(x) \delta + \lambda(\delta^2 - \beta)$$
$$\nabla_\delta L(\delta, \lambda) = \nabla f(x)^\top + \delta^\top (\nabla^2 f(x) + 2\lambda \mathbf{I})$$

  gives the step $\delta = -(\nabla^2 f(x) + 2\lambda \mathbf{I})^{-1} \nabla f(x)$, with λ the **dual variable**

- For $\lambda \to \infty$, δ becomes aligned with $-\nabla f(x)$ (but $|\delta| \to 0$)

Newton method with non-pos-def fallback

initialize stepsize $\alpha = 1$
repeat
  try to compute δ to solve $(\nabla^2 f(x) + \lambda \mathbf{I})\ \delta = -\nabla f(x)$
  if $\nabla f(x)^\top \delta > 0$ (non-descent) or fails (ill-def. linear system) **then**
    $\delta \leftarrow -\frac{\nabla f(x)}{|\nabla f(x)|}$ *// (gradient direction)*
    (Or: choose $\lambda > [-\text{minimal eigenvalue of } \nabla^2 f(x)]^+$ and repeat)
  end if
  while $f(x + \alpha\delta) > f(x) + \varrho_{\text{ls}} \nabla f(x)^\top (\alpha\delta)$ **do** *// line search*
    $\alpha \leftarrow \varrho_\alpha^- \alpha$ *// decrease stepsize*
    optionally: $\lambda \leftarrow \varrho_\lambda^+ \lambda$ and recompute d *// increase damping*
  end while
  $x \leftarrow x + \alpha\delta$ *// step is accepted*
  $\alpha \leftarrow \min\{\varrho_\alpha^+ \alpha, 1\}$ *// increase stepsize*
  optionally: $\lambda \leftarrow \varrho_\lambda^- \lambda$ *// decrease damping*
until $\|\alpha\delta\|_\infty < \theta$ repeatedly

- See comments in exercise e02
- Demo ...

- In the remainder: Extensions of the Newton approach:
  - Gauss-Newton
  - Quasi-Newton
  - BFGS, (L)BFGS
  - Conjugate Gradient

- And a crazy method: Rprop

Gauss-Newton method

- Consider a **sum-of-squares** problem:

$$\min_x f(x) \qquad \text{where } f(x) = \phi(x)^\top \phi(x) = \sum_{i=1}^{d} \phi_i(x)^2$$

with **features** $\phi(x) \in \mathbb{R}^d$, and we can evaluate $\phi(x)$, $\frac{\partial}{\partial x}\phi(x)$ for any $x \in \mathbb{R}^n$

- $\phi(x) \in \mathbb{R}^d$ is a vector; each entry contributes a squared cost term to $f(x)$
- $\frac{\partial}{\partial x}\phi(x)$ is the **Jacobian** $(d \times n$-matrix)

$$\frac{\partial}{\partial x}\phi(x) = \begin{pmatrix} \frac{\partial}{\partial_{x_1}}\phi_1(x) & \frac{\partial}{\partial_{x_2}}\phi_1(x) & \cdots & \frac{\partial}{\partial_{x_n}}\phi_1(x) \\ \frac{\partial}{\partial_{x_1}}\phi_2(x) & & & \vdots \\ \vdots & & & \vdots \\ \frac{\partial}{\partial_{x_1}}\phi_d(x) & \cdots & \cdots & \frac{\partial}{\partial_{x_n}}\phi_d(x) \end{pmatrix} \in \mathbb{R}^{d \times n}$$

with 1st-order Taylor expansion $\phi(x+\delta) = \phi(x) + \frac{\partial}{\partial x}\phi(x)\,\delta$

Gauss-Newton method

- The gradient and Hessian of $f(x)$ are

$$
\begin{aligned}
f(x) &= \phi(x)^\top \phi(x) \\
\nabla f(x) &= 2\frac{\partial}{\partial x}\phi(x)^\top \phi(x) \qquad \text{(recall } \nabla f(x) \equiv \frac{\partial}{\partial x} f(x)^\top\text{)} \\
\nabla^2 f(x) &= 2\frac{\partial}{\partial x}\phi(x)^\top \frac{\partial}{\partial x}\phi(x) + 2\phi(x)^\top \nabla^2 \phi(x)
\end{aligned}
$$

- *The Gauss-Newton method is the Newton method for* $f(x) = \phi(x)^\top \phi(x)$ *while approximating* $\nabla^2 \phi(x) \approx 0$

  In the Newton algorithm, replace line 3 by solving

$$
(2\frac{\partial}{\partial x}\phi(x)^\top \frac{\partial}{\partial x}\phi(x) + \lambda \mathbf{I})\ \delta = -2\frac{\partial}{\partial x}\phi(x)^\top \phi(x)
$$

Gauss-Newton method

- The approximate Hessian $H = 2\frac{\partial}{\partial x}\phi(x)^\top \frac{\partial}{\partial x}\phi(x)$ is **always semi-pos-def!**

Gauss-Newton method

- The approximate Hessian $H = 2\frac{\partial}{\partial x}\phi(x)^\top \frac{\partial}{\partial x}\phi(x)$ is **always semi-pos-def!**

- H is a sum of rank-1 matrices:

$$H = 2\sum_{i=1}^{d} \nabla\phi_i(x)^\top \nabla\phi_i(x)$$

  (which implies semi-pos-def)

Gauss-Newton method

- The approximate Hessian $H = 2\frac{\partial}{\partial x}\phi(x)^\top\frac{\partial}{\partial x}\phi(x)$ is **always semi-pos-def!**

- H is a sum of rank-1 matrices:

$$H = 2\sum_{i=1}^{d} \nabla\phi_i(x)^\top \nabla\phi_i(x)$$

  (which implies semi-pos-def)

- Computing H requires only first-order derivatives of features ϕ, no computationally expensive Hessians

Gauss-Newton method

- The approximate Hessian $H = 2\frac{\partial}{\partial x}\phi(x)^\top \frac{\partial}{\partial x}\phi(x)$ is **always semi-pos-def!**

- H is a sum of rank-1 matrices:

$$H = 2\sum_{i=1}^{d} \nabla\phi_i(x)^\top \nabla\phi_i(x)$$

  (which implies semi-pos-def)

- Computing H requires only first-order derivatives of features ϕ, no computationally expensive Hessians

- H can be interpreted as pullback of Euclidean norm $\phi^\top\phi$ in feature space. As it is x-dependent, this is a non-constant metric in x-space – it defines a *Riemannian* metric. (See math notes.)

Robotics example

- Sum-of-squares problems appear at many places in AI: "minimize squared errors"

- Basic robotics example: Trajectory optimization
  Let $q : \{1, .., T\} \mapsto \mathbb{R}^n$ be a discretized path in $\mathbb{R}^n$

$$\min_q \sum_{t=1}^{T} (q_t + q_{t\text{-}2} - 2q_{t\text{-}1})^2 \ + \ \phi(q_T)^2$$

  where $q_0, q_{\text{-}1}$ are given, and $\phi(q_T)$ are some features of the end configuration q_T

Quasi-Newton methods

Quasi-Newton methods

- Assume we *cannot* evaluate $\nabla^2 f(x)$.

  Can we still use 2nd order methods?

- Yes: We can approximate $\nabla^2 f(x)$ from the data $\{(x_i, \nabla f(x_i))\}_{i=1}^k$ of previous iterations

  (General view: We can *learn* from the data $\{(x_i, \nabla f(x_i))\}_{i=1}^k$ $\rightsquigarrow$ e.g., Bayesian optimization or model-based for blackbox optimization)

Basic example

- We've seen already two data points $(x_1, \nabla f(x_1))$ and $(x_2, \nabla f(x_2))$
  How can we estimate $\nabla^2 f(x)$?

- In 1D:

$$\nabla^2 f(x) \approx \frac{\nabla f(x_2) - \nabla f(x_1)}{x_2 - x_1}$$

- In $\mathbb{R}^n$: let $y = \nabla f(x_2) - \nabla f(x_1),\ \delta = x_2 - x_1$

  Find "simplest" approximate Hessian matrix H or $H^{\text{-}1}$ to fulfil

$$H\ \delta \overset{!}{=} y \qquad \text{or} \qquad \delta \overset{!}{=} H^{\text{-}1} y \tag{1}$$

  (The first equation is called *secant equation*)

Basic example

- We've seen already two data points $(x_1, \nabla f(x_1))$ and $(x_2, \nabla f(x_2))$
  How can we estimate $\nabla^2 f(x)$?

- In 1D:

$$\nabla^2 f(x) \approx \frac{\nabla f(x_2) - \nabla f(x_1)}{x_2 - x_1}$$

- In $\mathbb{R}^n$: let $y = \nabla f(x_2) - \nabla f(x_1)$, $\delta = x_2 - x_1$

  Find "simplest" approximate Hessian matrix H or H^{-1} to fulfil

$$H\,\delta \overset{!}{=} y \qquad \text{or} \qquad \delta \overset{!}{=} H^{-1} y \tag{1}$$

  (The first equation is called *secant equation*)

- Symmetric rank-1 solutions for H and H^{-1}:

$$H = \frac{yy^\top}{y^\top \delta} \qquad \text{or} \qquad H^{-1} = \frac{\delta\delta^\top}{\delta^\top y} \tag{2}$$

  [Left: how to update H. Right: how to update directly H^{-1}.]

BFGS

- Broyden-Fletcher-Goldfarb-Shanno (BFGS) method:

Input: initial $x \in \mathbb{R}^n$, functions $f(x)$, $\nabla f(x)$, tolerance θ
Output: x

initialize $H^{\text{-}1} = \mathbf{I}_n$
repeat
    compute $\delta = -H^{\text{-}1} \nabla f(x)$
    perform a line search $\min_\alpha f(x + \alpha\delta)$
    $\delta \leftarrow \alpha\delta$
    $y \leftarrow \nabla f(x + \delta) - \nabla f(x)$
    $x \leftarrow x + \delta$
    update $H^{\text{-}1} \leftarrow \left(\mathbf{I} - \frac{y\delta^\top}{\delta^\top y}\right)^\top H^{\text{-}1} \left(\mathbf{I} - \frac{y\delta^\top}{\delta^\top y}\right) + \frac{\delta\delta^\top}{\delta^\top y}$
until $\|\delta\|_\infty < \theta$

- Notes:
  - The blue term is the $H^{\text{-}1}$-update as on the previous slide
  - The red term "deletes" previous $H^{\text{-}1}$-components. Check: $H^{\text{-}1} y = \delta$
  - equivalent to the Sherman-Morrison formula: $H \leftarrow H - \frac{H\delta\delta^\top H^\top}{\delta^T H\delta} + \frac{yy^\top}{y^\top\delta}$

L-BFGS

- In high dimensions, we do not want to explicitly store a dense H^{-1}. Instead we store vectors $\{v_i\}$ such that $H^{-1} = \sum_i v_i v_i^\top$
- **L-BFGS** (limited memory BFGS) limits the rank of the H^{-1} and thereby the used memory (number of vectors v_i)

L-BFGS

- In high dimensions, we do not want to explicitly store a dense H^{-1}. Instead we store vectors $\{v_i\}$ such that $H^{-1} = \sum_i v_i v_i^\top$
- **L-BFGS** (limited memory BFGS) limits the rank of the H^{-1} and thereby the used memory (number of vectors v_i)

- Some thoughts:
  In principle, there are alternative ways to estimate H^{-1} from the data $\{(x_i, f(x_i), \nabla f(x_i))\}_{i=1}^k$, e.g. using Gaussian Process regression with derivative observations
  – not only the derivatives but also the value $f(x_i)$ should give information on $H(x)$ for non-quadratic functions
  – should one weight 'local' data stronger than 'far away'? (GP covariance function)
  – related to model-based search (see Blackbox Optimization lecture)

(Nonlinear) Conjugate Gradient

Conjugate Gradient

- The "Conjugate Gradient Method" is a method for solving (large, or sparse) linear eqn. systems $Ax + b = 0$, without inverting or decomposing A. The steps will be "A-orthogonal" (=conjugate). We mention its extension for optimizing nonlinear functions $f(x)$

- As before we evaluted $g' = \nabla f(x_1)$ and $g = \nabla f(x_2)$ at two different points $x_1, x_2 \in \mathbb{R}^n$
- Additional assumption: *exact line-search* step to x_2:
  - $x_2 = x_1 + \alpha\delta_1 \ , \quad \alpha = \text{argmin}_\alpha \, f(x_1 + \alpha\delta_1)$
  - iso-lines of $f(x)$ at x_2 are tangential to δ_1

$\Rightarrow$ The next search direction should be "orthogonal" to the previous one, but orthogonal w.r.t. the Hessian H, i.e., $d_2^\top H d_1 = 0$, which is called conjugate

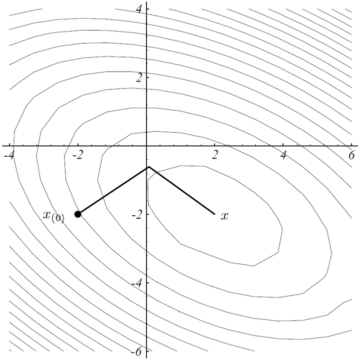
Conjugate Gradient

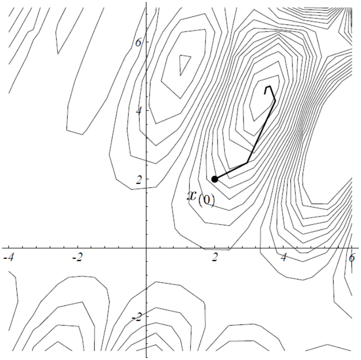
Input: initial $x \in \mathbb{R}^n$, functions $f(x), \nabla f(x)$, tolerance θ
Output: x

```
initialize descent direction d = g = −∇f(x)
repeat
    α ← argmin_α f(x + αd)                       // line search
    x ← x + αd
    g' ← g,  g = −∇f(x)                          // store and compute grad
    β ← max{ g^⊤(g − g') / (g'^⊤ g'), 0 }
    d ← g + βd                                   // conjugate descent direction
until |Δx| < θ
```

- Notes:
  – $\beta > 0$: The new descent direction always adds a bit of the old direction!
  – This *momentum* essentially provides 2nd order information
  – The equation for β is by Polak-Ribière: On a quadratic function $f(x) = x^\top A x + b^\top x$ this leads to **conjugate** search directions, $d'^\top A d = 0$.

Conjugate Gradient





- For quadratic functions CG converges in n iterations. But each iteration does *line search*

Rprop

Rprop

"Resilient Back Propagation" (outdated name from NN times...)

Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $\nabla f(x)$, initial stepsize α, tolerance θ
Output: x

initialize $x = x_0$, all $\alpha_i = \alpha$, all $g_i = 0$
repeat
  $g \leftarrow \nabla f(x)$
  $x' \leftarrow x$
  for $i = 1 : n$ **do**
    if $g_i g'_i > 0$ **then** *// same direction as last time*
      $\alpha_i \leftarrow 1.2 \alpha_i$
      $x_i \leftarrow x_i - \alpha_i \ \text{sign}(g_i)$
      $g'_i \leftarrow g_i$
    else if $g_i g'_i < 0$ **then** *// change of direction*
      $\alpha_i \leftarrow 0.5 \alpha_i$
      $x_i \leftarrow x_i - \alpha_i \ \text{sign}(g_i)$
      $g'_i \leftarrow 0$ *// force last case next time*
    else
      $x_i \leftarrow x_i - \alpha_i \ \text{sign}(g_i)$
      $g'_i \leftarrow g_i$
    end if
    optionally: cap $\alpha_i \in [\alpha_{\text{min}}\, x_i, \alpha_{\text{max}}\, x_i]$
  end for
until $|x' - x| < \theta$ for 10 iterations in sequence

Rprop

- Rprop is a bit crazy:
  - stepsize adaptation in each dimension *separately*
  - it not only ignores $|\nabla f|$ but also its exact direction
    step directions may differ up to $< 90^\circ$ from ∇f
  - Often works very robustly
  - Guarantees? See work by Ch. Igel

- If you like, have a look at:
  Christian Igel, Marc Toussaint, W. Weishui (2005): Rprop using the natural gradient compared to Levenberg-Marquardt optimization. In Trends and Applications in Constructive Approximation. International Series of Numerical Mathematics, volume 151, 259-272.

Bound Constrained Optimization – postponed

- Bound constrained optimization:

$$\min_x f(x) \quad \text{s.t.} \quad l_i \le x_i \le u_i$$

  simple box constraints on the variables (“simple constraints”)

- It might seem straight-forward to extend methods, perhaps by just clipping steps – but turns out not at all straight-forward
  $\to$ we'll discuss it after constrained optimization

  (e.g., Bertsekas: “Projected Newton methods for optimization problems with simple constraints”)

Appendix

Stopping Criteria

- Standard references (Boyd) define stopping criteria based on the "change" in $f(x)$, e.g. $|\Delta f(x)| < \theta$ or $|\nabla f(x)| < \theta$.

- Throughout I will define stopping criteria based on the change in x, e.g. $|\Delta x| < \theta$ repeatedly. In my experience with certain applications this is more meaningful, and invariant to the scaling of f. But this is application dependent.

Evaluating optimization costs

- Standard references (Boyd) assume line search is cheap and measure optimization costs as the number of iterations (counting 1 per line search).

- Throughout I will assume that every evaluation of $f(x)$ or $(f(x), \nabla f(x))$ or $(f(x), \nabla f(x), \frac{\partial}{\partial x}\phi(x))$ is approx. equally expensive—as is the case in certain applications.