Optimization

Find maxima/minima of real-valued $f(\vec{x})$ over a set $S \subset \mathbb{R}^n$

$$\min_{\vec{x}\in S} f(\vec{x}) \qquad \vec{x}^* = \arg\min_{\vec{x}\in S} f(\vec{x}) = \text{minimizer}$$

$f(\vec{x})$ is called the objective or cost or loss function

$\vec{x}$ is the state variable, S is the admissible or feasible set

Types of optimization problems:

global - local

unconstrained - constrained

linear programming - nonlinear - quadratic - integer

convex - nonconvex

combinatorial: network optimization

Optimization Methods

1. 1st order (gradient): (local) Gradient Descent (GD), SGD = Stochastic Gradient Descent in ML
   Conjugate Gradient (CG)
   subgradient
2. 2nd order: Newton type: Newton: solve $\nabla f = 0$
   QuasiNewton
   truncated Newton
   Marquardt-Levenberg (Gnuplot uses it in 'fit')
3. Derivative free: Trust region
   Nelder-Mead (1965): one of the best, simplex based
   Monte-Carlo, Stochastic tunneling, ...
4. Combinatorial: Simulated Annealing
   Genetic Algorithms
   Particle Swarm
   ...

Optimization is huge area, very diverse, very mathematical

Operations Research ($\rightarrow$ Industrial Engineering)
Management Science, Control Engineering, Control Theory,
Optimal Control, Calculus of Variations, ..., Machine Learning!

Optimization problems arise everywhere
in Science, Technology, Business, Math, ...

Global min if $f(x^*) \leq f(x) \quad \forall x \in S$

Local min if $f(x^*) \leq f(x)$ for x in a neighborhood of x^*

Finding global min is very hard in high dimensions
possible only for "nice" f (e.g. as in linear programming)

Best we can hope for is finding local min, which sometimes is good enough...

Basic ideas/approaches come from 1-dim Calculus:

extrema are critical points (necessary but not sufficient)
(may be saddle pt)
or boundary pts in constrained problems

Critical (stationary) points: when $f'(x)=0$ or f' does not exist
are candidates only, must be checked...

in $\mathbb{R}^n$: $\nabla f(\vec{x}) = \vec{0}$ or ∇f does not exist
system of nonlinear equs must be solved

Sufficient conditions: $f''(x_c) > 0 \Rightarrow x_c$ is local min
$< 0 \Rightarrow$ " " " max
$= 0 \Rightarrow$ " " inflection (saddle) pt

in $\mathbb{R}^n$: Hessian of f: $H = \underbrace{[H_{ij}]}_{n \times n \text{ matrix}} = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]$ pos. definite $\Rightarrow$ local min
neg. " $\Rightarrow$ local max
indefinite $\Rightarrow$ saddle pt

H positive definite means $H\xi\cdot\xi > 0 \quad \forall \xi \in \mathbb{R}^n$ (hard to check)
$\Leftrightarrow$ all eigenvalues > 0 (expensive!)

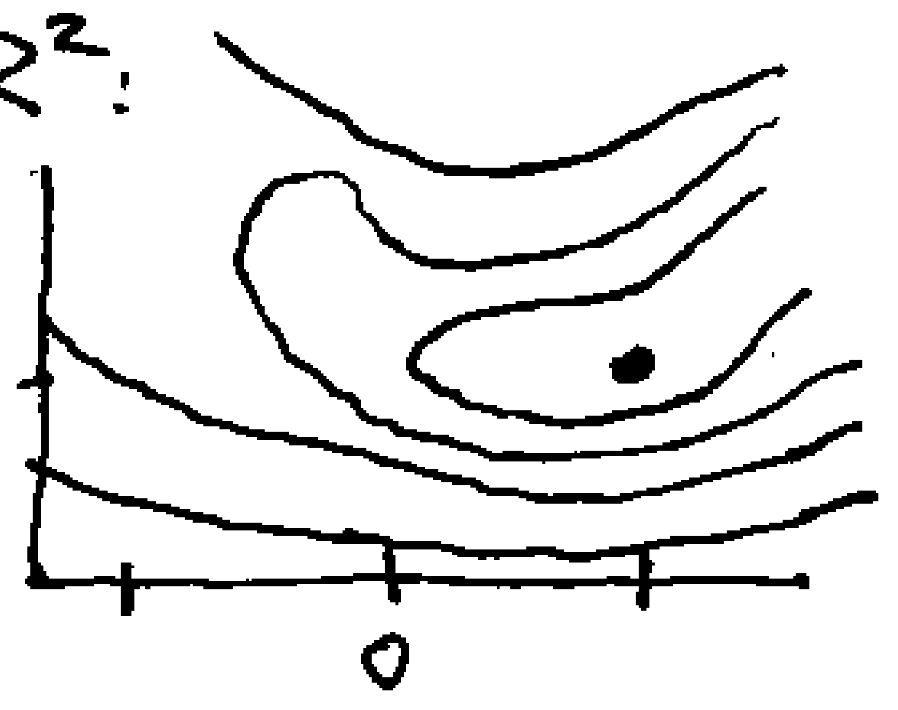
At a critical pt x_c $f(x_c+h) \approx f(x_c) + 0 + \frac{1}{2}h^T H h + \ldots$ so f resembles a quadratic near x_c

Unconstrained Optimization: $\min_{\vec{x} \in \mathbb{R}^n} f(\vec{x})$

Tough example/test problem: Rosenbrock function, in $\mathbb{R}^2$:

$$\min f(x_1, x_2) = (1-x_1)^2 + 100\,(x_2 - x_1^2)^2$$

has min at $(1,1)$, but very hard to compute!



crit. pts: $g_1 = \frac{\partial f}{\partial x_1} = -2(1-x_1) + 200\,(x_2 - x_1^2)(-2x_1) = 0$

$g_2 = \frac{\partial f}{\partial x_2} = 200\,(x_2 - x_1^2) \qquad = 0$

$$H = \begin{bmatrix} 2 + 1200x_1^2 - 400x_2 & -x_1 \cdot 400 \\ -400x_1 & 200 \end{bmatrix}, \quad H(1,1) = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$$

eigenvalues $\lambda_1 = 1001.4$, $\lambda_2 = 0.4$

contours very elongated in only one direction

Basic methods for 1-dim optimization: $\min_t f(t)$, $f: \mathbb{R} - \mathbb{R}$

1. Golden Section search: for "unimodel" f; brackets min, safe, only lin. convergent

2. Parabolic Interpolation: fit a parabola thru 3 values, min parabola ← next iter

   superlinear convergence rate ≈ 1.324

3. Newton: approximate f by a quadratic: $f(t+h) \approx f(t) + f'(t) \cdot h + \frac{f''(t)}{2} h^2 = q$

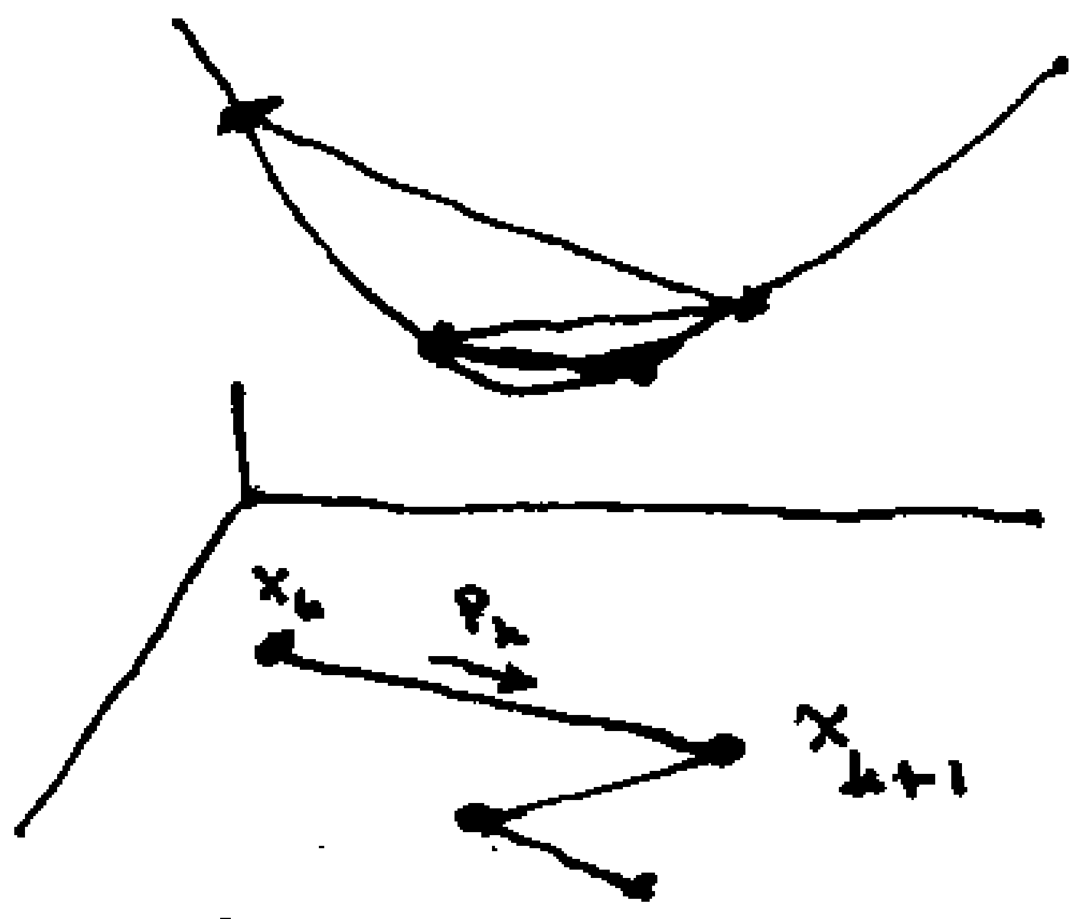
   min $q(h)$: $q'(h) = f'(t) + f''(t) \cdot h = 0 \Rightarrow h = -\frac{f'(t)}{f''(t)}$, $t_{new} = t + h$

   Clearly this is Newton-Raphson for $f'(t) = 0$.

   Converges quadratically but must start very close to local min

n-dim Optimization: $\min_{\vec{x}} f(\vec{x})$, $f: \mathbb{R}^n \to \mathbb{R}$

Local search methods



Start with a guess $\vec{x}_0$. For $k = 0, 1, 2, \ldots$

1. compute a search direction $\vec{p}_k$
2. compute a step length t_k
3. $\vec{x}_{k+1} = \vec{x}_k + t_k \vec{p}_k$

Want $\vec{p}_k$ to be a direction along which f decreases, a descent direction

$$f(\vec{x}_k + t\vec{p}_k) - f(\vec{x}_k) \approx t \, \nabla f(\vec{x}_k) \cdot \vec{p}_k \overset{\text{want}}{<} 0$$

want directional derivative $\nabla f \cdot p_k < 0$

Many methods for choosing search (descent) direction $\vec{p}_k$:

non-derivative methods, gradient methods, 2nd deriv. (Newton) methods

Once $\vec{p}_k$ is chosen, a linesearch method tries to find a step length t_k

to $\min_t f(\vec{x}_k + t\vec{p}_k)$ along $\vec{p}_k$ which is 1-dim minimization

Fundamental Methods:

Gradient Methods: Steepest Descent Methods (Gradient Descent in ML)
Conjugate Gradient Methods

Newton Methods: Newton
Quasi-Newton
Truncated Newton

Gradient Descent

1. Steepest Descent: choose $\vec{p}_k = -\vec{g}_k := -\nabla f(x_k)$ = direction of steepest local descent of f at $\vec{x}_k$

choose t_k by $\min_t f(x_k - t g_k)$ by some line search method

It is simplest, oldest, reliable, makes progress when far from minimum but slows down (to linear rate with factor ≈ 1!) when close to a local min. Great for starting off other (faster) methods. In ML: SGD, ADAM

2. Newton Method: solve $\nabla f(\vec{x}) = \vec{0}$ by Newton (now $f'' \to$ Hessian matrix)

$$x_{k+1} = x_k - H_k^{-1} \nabla f(x_k), \quad H_k = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]\Big|_{\vec{x}_k}$$

Of course we don't invert H_k! we just solve the linear system

$$H_k s = -\nabla f(x_k) \text{ for } s, \text{ then } x_{k+1} = x_k + s$$

Again quadratic convergence when near min, so use a search dir method to get "close", or use a Trust Region Method (estimate a radius about x_k within which the quadratic approx. is good enough for Newton.

Big disadvantage: Hessian must be computed: must be available, and expensive to evaluate!
Various alternatives:

3. Quasi Newton Methods: $x_{k+1} = x_k - t_k B_k^{-1} \nabla f(x_k)$

t_k: a line search parameter

B_k: some approximation to H_k:
- secant updating, finite differences,
- freeze H_k for a few iterations, then re-evaluate
- neglect some terms in H_k

...

These methods are more robust, much cheaper than Newton, superlinear convergent

4. Conjugate Gradient Methods: use a modified gradient:

Start with a guess x_0, set $g_0 = -\nabla f(x_0)$, $p_0 = -g_0$, then iterate n times:

1. $x_{k+1} = x_k + t_k p_k$, t_k from a line search method
2. $g_{k+1} = \nabla f(x_{k+1})$
3. $\gamma_{k+1} = \dfrac{g_{k+1} \cdot g_{k+1}}{g_k \cdot g_k}$ (Fletcher-Reeves)

   or $\gamma_{k+1} = \dfrac{(g_{k+1} - g_k) \cdot g_{k+1}}{g_k \cdot g_k}$ (Polak-Ribiere), or.... (Hestenes-Stiefel)

4. $p_{k+1} = -g_{k+1} + \gamma_{k+1} p_k$

γ's come from nice geometric ideas...

CG methods arose from solving linear systems $Ax = b$, with A symm. pos. definite (like a Hessian)
via minimization of the quadratic form $q(x) = \frac{1}{2} Ax \cdot x - b \cdot x$
which is an equivalent problem: $\nabla q = \frac{1}{2}(\nabla \cdot Ax)x + \frac{1}{2} Ax(\nabla \cdot x) - b(\nabla \cdot x)$

$$= (\nabla \cdot x)\left[\tfrac{1}{2}Ax + \tfrac{1}{2}Ax - b\right] = (\nabla \cdot x)\left[Ax - b\right] = 0 \Leftrightarrow Ax = b!$$

Nice geometric ideas lead to choosing $p \perp \nabla q$ w.r.t. $\langle u,v\rangle := \langle Au, v\rangle$
Method converges to $\min q$ in n iters theoretically in exact arithmetic, but roundoff can destroy $\perp$. Was neglected for many years as impractical, till 1967 when J.D. Evans showed that "preconditioning" A appropriately speeds it up significantly!
Preconditioning became big in all sorts of problems, PCG became great solver for $Ax=b$, extended, generalized, and such modifications imported back to optimization!
The γ's are such that if f is a convex quadratic and t_k are exact minimizers then the nonlinear CG reduces to linear CG for $Ax=b$ and terminates in $\leq n$ iters.
Great advantages of CG-type methods: storage, computation of order n only, much better convergence than Steepest Descent. Widely used in min and for $Ax=b$, and it is basis for Quasi-Newton methods.

Minimization and system solvers : many connections, both ways

$Ax=b$ for A symm. pos. definite $\Leftrightarrow$ $\min q(x)$, $q(x)=\frac{1}{2}Ax\cdot x - b\cdot x$ $\Rightarrow$ CG methods

$Ax=b$ for any A : $r=b-Ax$, $\min_x \frac{1}{2}\|r\|_2^2 = \min_x \frac{1}{2} r^T r =: \min_x f(x)$

normal eqs: $\nabla f = 0 \Leftrightarrow A^T A x = A^T b$ handled best via orthogonalization

$\vec{F}(\vec{x}) = \vec{0}$ aquivalent to $\min q(\vec{x})$, $q(\vec{x}) = \frac{1}{2}\|\vec{F}(\vec{x})\|_2^2 = \frac{1}{2}\sum_{i=1}^{n} |F_i|^2$ whose min is $\vec{0}$

So, min solvers can be used a nonlinear system solvers

Conversely, minimizing f via critical pts means solving $\nabla f = \vec{0}$

so, systen solvers (like Newton) can be used for $\min f$, as discussed.

Linear Least Squares for data (t_i, y_i), $i=1:m$: model $\Phi(t, \vec{x}) = \sum_{k=1}^{n} x_k \varphi_k(t)$, basis $\{\varphi_k\}$, $\overset{\vec{c}}{\overset{\|}{\vec{x}}}$

$\min_{\vec{x}} f(\vec{x})$, $f(\vec{x}) = E(\vec{x}) = \sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} x_k \varphi_k(t_i)\right)^2$ = LS error = R^2 error

normal eqs: $\nabla f(\vec{x}) = \vec{0}$ is linear system $Ax \overset{LS}{\approx} b$, A $m\times n$

Set $r = b - Ax$, $\min \frac{1}{2} r^T r$, $\nabla f = 0 \Leftrightarrow A^T A x = A^T b$, $\vec{b} = \vec{y}$

Nonlinear Least Squares for data (t_i, y_i), $i=1:m$: model $\Phi(t, \vec{x})$ any

$\min_x f(\vec{x})$, $f(\vec{x}) = E(\vec{x}) = \sum_{i=1}^{m}\left(y_i - \Phi(t_i, \vec{x})\right)^2$ = LS error

Set $\vec{F}(\vec{x}) = \vec{y} - \vec{\Phi}(\vec{t}, \vec{x})$, so $f(\vec{x}) = \|\vec{F}(\vec{x})\|_2^2 = F^T F$

$\nabla f = J^T F$, J = Jacobian of $\vec{F}$

Marquard-Levenberg : one of the most robust and effective methods:

$\vec{x}_{k+1} = \vec{x}_k + \vec{s}_k$, with $\vec{s}_k$ solution of

$$\left[J^T(\vec{x}_k) J(\vec{x}_k) + \mu_k I\right]\vec{s}_k \approx -J^T(\vec{x}_k)\vec{F}(\vec{x}_k)$$

solved in Least Square sense :

$$\begin{bmatrix} J(\vec{x}_k) \\ \sqrt{\mu_k}\, I \end{bmatrix} \vec{s}_k \approx \begin{bmatrix} -\vec{F}(\vec{x}_k) \\ 0 \end{bmatrix}$$