Trajectory-based optimization

Emo Todorov

Applied Mathematics and Computer Science & Engineering

University of Washington

Winter 2012

Using the maximum principle

Recall that for deterministic dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ and cost rate $\ell(\mathbf{x}, \mathbf{u})$ the optimal state-control-costate trajectory $(\mathbf{x}(\cdot), \mathbf{u}(\cdot), \boldsymbol{\lambda}(\cdot))$ satisfies

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ -\dot{\boldsymbol{\lambda}} &= \ell_{\mathbf{x}}(\mathbf{x}, \mathbf{u}) + \mathbf{f}_{\mathbf{x}}(\mathbf{x}, \mathbf{u})^{\mathsf{T}} \boldsymbol{\lambda} \\ \mathbf{u} &= \arg\min_{\widetilde{\mathbf{u}}} \left\{ \ell(\mathbf{x}, \widetilde{\mathbf{u}}) + \mathbf{f}(\mathbf{x}, \widetilde{\mathbf{u}})^{\mathsf{T}} \boldsymbol{\lambda} \right\} \end{aligned}$$

with $\mathbf{x}(0)$ given and $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x}(T))$. Solving this boundary-value ODE problem numerically is a trajectory-based method.

Using the maximum principle

Recall that for deterministic dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ and cost rate $\ell(\mathbf{x}, \mathbf{u})$ the optimal state-control-costate trajectory $(\mathbf{x}(\cdot), \mathbf{u}(\cdot), \boldsymbol{\lambda}(\cdot))$ satisfies

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ -\dot{\boldsymbol{\lambda}} &= \ell_{\mathbf{x}}(\mathbf{x}, \mathbf{u}) + \mathbf{f}_{\mathbf{x}}(\mathbf{x}, \mathbf{u})^{\mathsf{T}} \boldsymbol{\lambda} \\ \mathbf{u} &= \arg\min_{\tilde{\mathbf{u}}} \left\{ \ell(\mathbf{x}, \tilde{\mathbf{u}}) + \mathbf{f}(\mathbf{x}, \tilde{\mathbf{u}})^{\mathsf{T}} \boldsymbol{\lambda} \right\} \end{aligned}$$

with $\mathbf{x}(0)$ given and $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_{\mathcal{T}}(\mathbf{x}(T))$. Solving this boundary-value ODE problem numerically is a trajectory-based method.

We can also use the fact that, if $(\mathbf{x}(\cdot), \boldsymbol{\lambda}(\cdot))$ satisfies the ODE for some $\mathbf{u}(\cdot)$ which is not a minimizer of the Hamiltonian $H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = \ell(\mathbf{x}, \mathbf{u}) + \mathbf{f}(\mathbf{x}, \mathbf{u})^{\mathsf{T}} \boldsymbol{\lambda}$, then the gradient of the total cost J is given by

$$\begin{aligned} J(\mathbf{x}(\cdot), \mathbf{u}(\cdot)) &= q_{\mathcal{T}}(\mathbf{x}(T)) + \int_0^T \ell(\mathbf{x}(t), \mathbf{u}(t))\, dt \\ \frac{\partial J}{\partial \mathbf{u}(t)} &= H_{\mathbf{u}}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = \ell_{\mathbf{u}}(\mathbf{x}, \mathbf{u}) + \mathbf{f}_{\mathbf{u}}(\mathbf{x}, \mathbf{u})^{\mathsf{T}} \boldsymbol{\lambda} \end{aligned}$$

Thus we can perform gradient descent on J with respect to $\mathbf{u}(\cdot)$

Compact representations

Given the current $\mathbf{u}(\cdot)$, each step of the algorithm involves computing $\mathbf{x}(\cdot)$ by integrating forward in time starting with the given $\mathbf{x}(0)$, then computing $\boldsymbol{\lambda}(\cdot)$ by integrating backward in time starting with $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_T(\mathbf{x}(T))$.

Compact representations

Given the current $\mathbf{u}(\cdot)$, each step of the algorithm involves computing $\mathbf{x}(\cdot)$ by integrating forward in time starting with the given $\mathbf{x}(0)$, then computing $\boldsymbol{\lambda}(\cdot)$ by integrating backward in time starting with $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_T(\mathbf{x}(T))$.

One way to implement the above methods is to discretize the time axis and represent $(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda})$ independently at each time step. This may be inefficient because the values at nearby time steps are usually very similar, thus it is a waste to represent/optimize them independently.

Compact representations

Given the current $\mathbf{u}(\cdot)$, each step of the algorithm involves computing $\mathbf{x}(\cdot)$ by integrating forward in time starting with the given $\mathbf{x}(0)$, then computing $\boldsymbol{\lambda}(\cdot)$ by integrating backward in time starting with $\boldsymbol{\lambda}(T) = \frac{\partial}{\partial \mathbf{x}} q_T(\mathbf{x}(T))$.

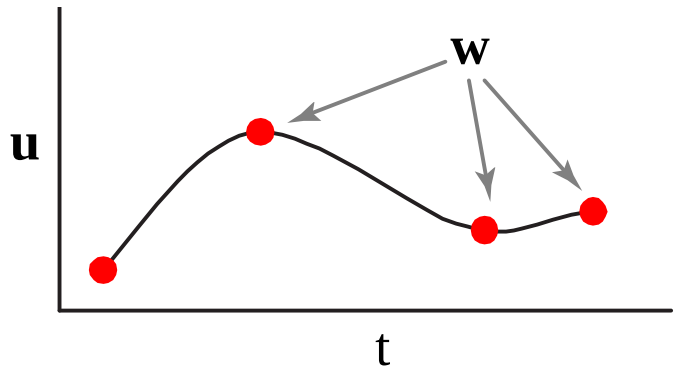
One way to implement the above methods is to discretize the time axis and represent $(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda})$ independently at each time step. This may be inefficient because the values at nearby time steps are usually very similar, thus it is a waste to represent/optimize them independently.

Instead we can splines, Legendre or Chebyshev polynomials, etc.

$$\mathbf{u}(t) = \mathbf{g}(t, \mathbf{w})$$

Gradient:

$$\frac{\partial J}{\partial \mathbf{w}} = \int_0^T \mathbf{g}_{\mathbf{w}}(t, \mathbf{w})^{\mathsf{T}} \frac{\partial J}{\partial \mathbf{u}(t)} dt$$



Space-time constraints

We can also minimize the total cost J as an explicit function of the (parameterized) state-control trajectory:

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{h}(t, \mathbf{v}) \\ \mathbf{u}(t) &= \mathbf{g}(t, \mathbf{w}) \end{aligned}$$

We have to make sure that the state-control trajectory is consistent with the dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$. This yields a constrained optimization problem:

$$\min_{\mathbf{v},\mathbf{w}} \left\{ q_{\mathcal{T}}(\mathbf{h}(T, \mathbf{v})) + \int_0^T \ell(\mathbf{h}(t, \mathbf{v}), \mathbf{g}(t, \mathbf{w}))\, dt \right\}$$

$$s.t. \quad \frac{\partial \mathbf{h}(t, \mathbf{v})}{\partial t} = \mathbf{f}(\mathbf{h}(t, \mathbf{v}), \mathbf{g}(t, \mathbf{v})), \quad \forall t \in [0, T]$$

Space-time constraints

We can also minimize the total cost J as an explicit function of the (parameterized) state-control trajectory:

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{h}(t, \mathbf{v}) \\ \mathbf{u}(t) &= \mathbf{g}(t, \mathbf{w}) \end{aligned}$$

We have to make sure that the state-control trajectory is consistent with the dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$. This yields a constrained optimization problem:

$$\min_{\mathbf{v},\mathbf{w}} \left\{ q_{\mathcal{T}}(\mathbf{h}(T, \mathbf{v})) + \int_0^T \ell(\mathbf{h}(t, \mathbf{v}), \mathbf{g}(t, \mathbf{w}))\, dt \right\}$$

$$s.t. \quad \frac{\partial \mathbf{h}(t, \mathbf{v})}{\partial t} = \mathbf{f}(\mathbf{h}(t, \mathbf{v}), \mathbf{g}(t, \mathbf{v})), \quad \forall t \in [0, T]$$

In practice we cannot impose the contraint for all t, so instead we choose a finite set of points $\{t_k\}$ where the constraint is enforced. The same points can also be used to approximate $\int \ell$. There may be no feasible solution (depending on $\mathbf{h}, \mathbf{g}$) in which case we have to live with constraint violations.

This requires no knowledge of optimal control (which may be why it is popular:)

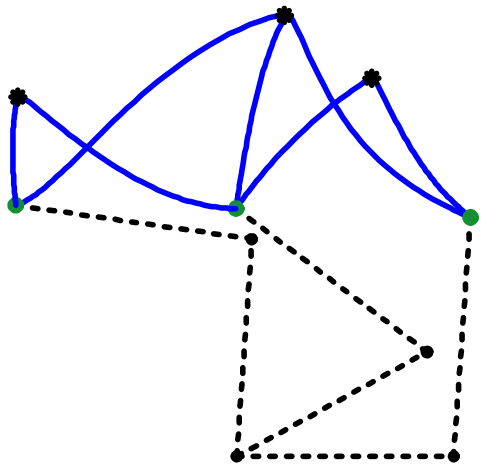
Second-order methods

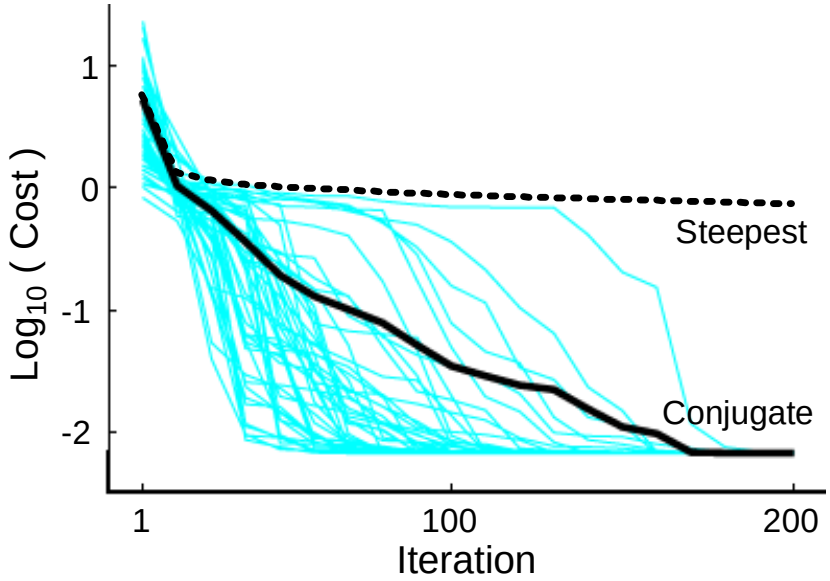
More efficient methods (DDP, iLQG) can be constructed by using the Bellman equations locally. Initialize with some open-loop control $\mathbf{u}^{(0)}(\cdot)$, and repeat:

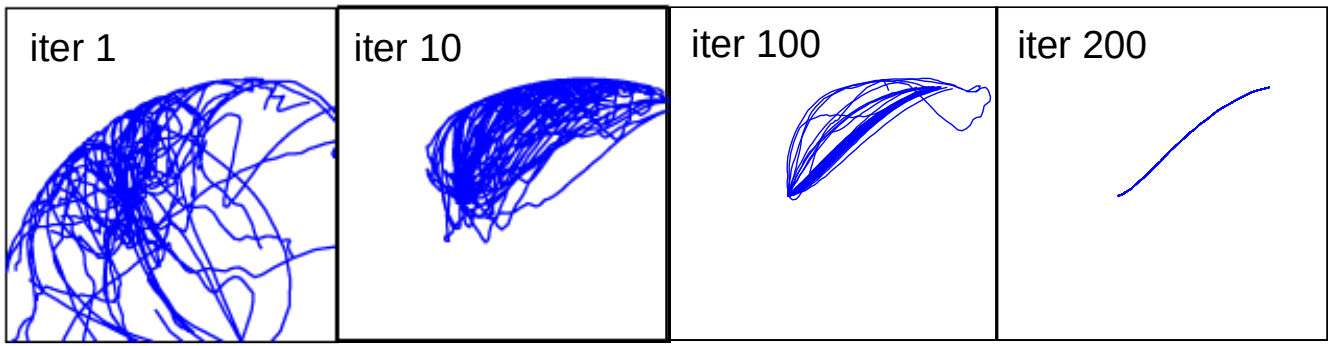
1. Compute the state trajectory $\mathbf{x}^{(n)}(\cdot)$ corresponding to $\mathbf{u}^{(n)}(\cdot)$.
2. Construct a time-varying linear (iLQG) or quadratic (DDP) approximation to the function $\mathbf{f}$ around $\mathbf{x}^{(n)}(\cdot), \mathbf{u}^{(n)}(\cdot)$, which gives the local dynamics in terms of the state and control deviations $\delta\mathbf{x}(\cdot), \delta\mathbf{u}(\cdot)$. Also construct quadratic approximations to the costs ℓ and $q_{\mathcal{T}}$.
3. Compute the locally-optimal cost-to-go $v^{(n)}(\delta\mathbf{x}, t)$ as a quadratic in $\delta\mathbf{x}$. In iLQG this is exact (because the local dynamics are linear and the cost is quadratic) while in DDP this is approximate.
4. Compute the locally-optimal linear feedback control law in the form $\boldsymbol{\pi}^{(n)}(\delta\mathbf{x}, t) = \mathbf{c}(t) - L(t)\,\delta\mathbf{x}$.
5. Apply $\boldsymbol{\pi}^{(n)}$ to the local dynamics (i.e. integrate forward in time) to compute the state-control modification $\delta\mathbf{x}^{(n)}(\cdot), \delta\mathbf{u}^{(n)}(\cdot)$, and set $\mathbf{u}^{(n+1)}(\cdot) = \mathbf{u}^{(n)}(\cdot) + \delta\mathbf{u}^{(n)}(\cdot)$. This requires linesearch to avoid jumping outside the region where the local approximation is valid.

Numerical comparison

	Conj.	ODE	DDP
time(s)	**5.9**	**1.25**	**0.61**







Gradient descent

The directional derivative of $f(\mathbf{x})$ at $\mathbf{x}_0$ in direction $\mathbf{v}$ is

$$D_{\mathbf{v}}[f](\mathbf{x}_0) = \left. \frac{df(\mathbf{x}_0 + \varepsilon \mathbf{v})}{d\varepsilon} \right|_{\varepsilon=0}$$

Let $\mathbf{x}(\varepsilon) = \mathbf{x}_0 + \varepsilon \mathbf{v}$. Then $f(\mathbf{x}_0 + \varepsilon \mathbf{v}) = f(\mathbf{x}(\varepsilon))$ and the chain rule yields

$$D_{\mathbf{v}}[f](\mathbf{x}_0) = \left. \frac{\partial \mathbf{x}(\varepsilon)^{\mathsf{T}}}{\partial \varepsilon} \right|_{\varepsilon=0} \left. \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{v}^{\mathsf{T}} \left. \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{v}^{\mathsf{T}} \mathbf{g}(\mathbf{x}_0)$$

where $\mathbf{g}$ denotes the gradient of f.

Gradient descent

The directional derivative of $f(\mathbf{x})$ at $\mathbf{x}_0$ in direction $\mathbf{v}$ is

$$D_{\mathbf{v}}[f](\mathbf{x}_0) = \left. \frac{df(\mathbf{x}_0 + \varepsilon \mathbf{v})}{d\varepsilon} \right|_{\varepsilon=0}$$

Let $\mathbf{x}(\varepsilon) = \mathbf{x}_0 + \varepsilon \mathbf{v}$. Then $f(\mathbf{x}_0 + \varepsilon \mathbf{v}) = f(\mathbf{x}(\varepsilon))$ and the chain rule yields

$$D_{\mathbf{v}}[f](\mathbf{x}_0) = \left. \frac{\partial \mathbf{x}(\varepsilon)^{\mathsf{T}}}{\partial \varepsilon} \right|_{\varepsilon=0} \left. \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{v}^{\mathsf{T}} \left. \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{v}^{\mathsf{T}} \mathbf{g}(\mathbf{x}_0)$$

where $\mathbf{g}$ denotes the gradient of f.

Theorem (steepest ascent direction)

The maximum of $D_{\mathbf{v}}[f](\mathbf{x}_0)$ s.t. $\|\mathbf{v}\| = 1$ is achieved when $\mathbf{v}$ is parallel to $\mathbf{g}(\mathbf{x}_0)$.

Gradient descent

The directional derivative of $f(\mathbf{x})$ at $\mathbf{x}_0$ in direction $\mathbf{v}$ is

$$D_{\mathbf{v}}[f](\mathbf{x}_0) = \left. \frac{df(\mathbf{x}_0 + \varepsilon \mathbf{v})}{d\varepsilon} \right|_{\varepsilon=0}$$

Let $\mathbf{x}(\varepsilon) = \mathbf{x}_0 + \varepsilon \mathbf{v}$. Then $f(\mathbf{x}_0 + \varepsilon \mathbf{v}) = f(\mathbf{x}(\varepsilon))$ and the chain rule yields

$$D_{\mathbf{v}}[f](\mathbf{x}_0) = \left. \frac{\partial \mathbf{x}(\varepsilon)^{\mathsf{T}}}{\partial \varepsilon} \right|_{\varepsilon=0} \left. \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{v}^{\mathsf{T}} \left. \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{v}^{\mathsf{T}} \mathbf{g}(\mathbf{x}_0)$$

where $\mathbf{g}$ denotes the gradient of f.

Theorem (steepest ascent direction)

The maximum of $D_{\mathbf{v}}[f](\mathbf{x}_0)$ s.t. $\|\mathbf{v}\| = 1$ is achieved when $\mathbf{v}$ is parallel to $\mathbf{g}(\mathbf{x}_0)$.

Algorithm (gradient descent)

Set $\mathbf{x}_{k+1} = \mathbf{x}_k - \beta_k \mathbf{g}(\mathbf{x}_k)$ where β_k is the step size. The optimal step size is

$$\beta_k^* = \arg\min_{\beta_k} f(\mathbf{x}_k - \beta_k \mathbf{g}(\mathbf{x}_k))$$

Line search

Most optimization methods involve an inner loop which seeks to minimize (or sufficiently reduce) the objective function constrained to a line: $f(\mathbf{x} + \varepsilon \mathbf{v})$, where $\mathbf{v}$ is such that a reduction in f is always possible for sufficiently small ε, unless f is already at a local minimum. In gradient descent $\mathbf{v} = -\mathbf{g}(\mathbf{x})$; other choices are possible (see below) as long as $\mathbf{v}^{\mathsf{T}} \mathbf{g}(\mathbf{x}) \leq 0$.

Line search

Most optimization methods involve an inner loop which seeks to minimize (or sufficiently reduce) the objective function constrained to a line: $f(\mathbf{x} + \varepsilon \mathbf{v})$, where $\mathbf{v}$ is such that a reduction in f is always possible for sufficiently small ε, unless f is already at a local minimum. In gradient descent $\mathbf{v} = -\mathbf{g}(\mathbf{x})$; other choices are possible (see below) as long as $\mathbf{v}^T \mathbf{g}(\mathbf{x}) \leq 0$.

This is called *linesearch*, and can be done in different ways:

1. Backtracking: try some ε, if $f(\mathbf{x} + \varepsilon \mathbf{v}) > f(\mathbf{x})$ reduce ε and try again.
2. Bisection: attempt to minimize $f(\mathbf{x} + \varepsilon \mathbf{v})$ w.r.t. ε using a bisection method.
3. Polysearch: attempt to minimize $f(\mathbf{x} + \varepsilon \mathbf{v})$ by fitting quadratic or cubic polynomials in ε, finding the minimum analytically, and iterating.

Exact minimization w.r.t. ε is often a waste of time because for $\varepsilon \neq 0$ the current search direction may no longer be a descent direction.

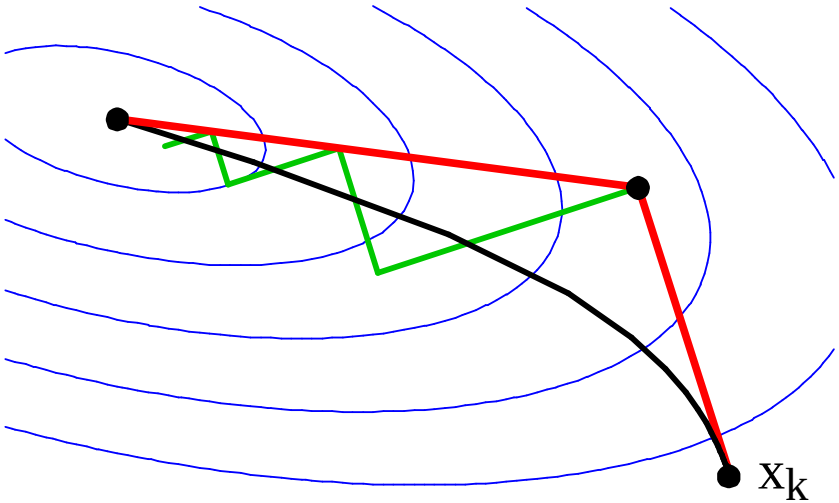
Sufficient reduction in f is defined relative to the local model (linear or quadratic). This is known as the Armijo-Goldstein condition; the Wolfe condition (which also involves the gradient) is more complicated.

Chattering

If $\mathbf{x}_{k+1}$ is a (local) minimum of f in the search direction $\mathbf{v}_k = -\mathbf{g}(\mathbf{x}_k)$, then $D_{\mathbf{v}_k}[f](\mathbf{x}_{k+1}) = 0 = \mathbf{v}_k^{\mathsf{T}} \mathbf{g}(\mathbf{x}_{k+1})$, and so if we use $\mathbf{v}_{k+1} = -\mathbf{g}(\mathbf{x}_{k+1})$ as the next search direciton, we have $\mathbf{v}_{k+1}$ orthogonal to $\mathbf{v}_k$. Thus gradient descent with exact line search (i.e. steepest descent) makes a 90 deg turn at each iteration, which causes chattering when the function has a long oblique valey.

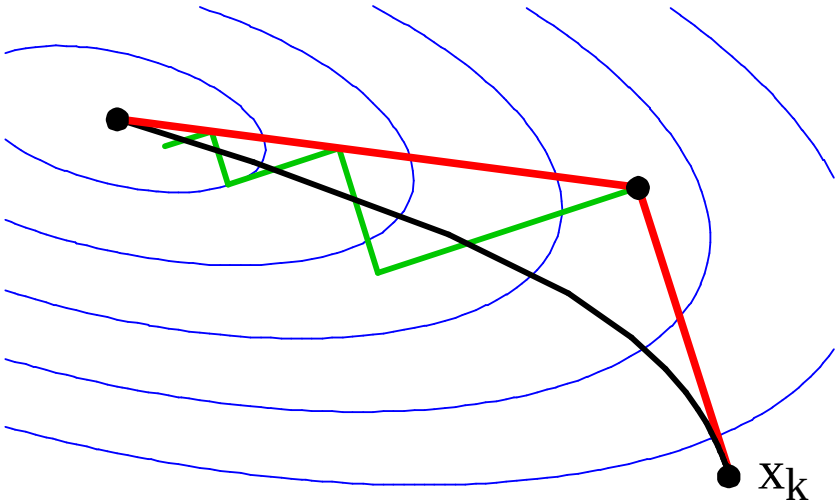
Chattering

If $\mathbf{x}_{k+1}$ is a (local) minimum of f in the search direction $\mathbf{v}_k = -\mathbf{g}(\mathbf{x}_k)$, then $D_{\mathbf{v}_k}[f](\mathbf{x}_{k+1}) = 0 = \mathbf{v}_k^T \mathbf{g}(\mathbf{x}_{k+1})$, and so if we use $\mathbf{v}_{k+1} = -\mathbf{g}(\mathbf{x}_{k+1})$ as the next search direciton, we have $\mathbf{v}_{k+1}$ orthogonal to $\mathbf{v}_k$. Thus gradient descent with exact line search (i.e. steepest descent) makes a 90 deg turn at each iteration, which causes chattering when the function has a long oblique valey.



Chattering

If $\mathbf{x}_{k+1}$ is a (local) minimum of f in the search direction $\mathbf{v}_k = -\mathbf{g}(\mathbf{x}_k)$, then $D_{\mathbf{v}_k}[f](\mathbf{x}_{k+1}) = 0 = \mathbf{v}_k^T \mathbf{g}(\mathbf{x}_{k+1})$, and so if we use $\mathbf{v}_{k+1} = -\mathbf{g}(\mathbf{x}_{k+1})$ as the next search direciton, we have $\mathbf{v}_{k+1}$ orthogonal to $\mathbf{v}_k$. Thus gradient descent with exact line search (i.e. steepest descent) makes a 90 deg turn at each iteration, which causes chattering when the function has a long oblique valey.



Key to developing more efficient methods is to anticipate how the gradient will rotate as we move along the current search direction.

Newton's method

Theorem

If all you have is a hammer, then everything looks like a nail.

Corollary

If all you can optimize is a quadratic, then every function looks like a quadratic.

Newton's method

Theorem

If all you have is a hammer, then everything looks like a nail.

Corollary

If all you can optimize is a quadratic, then every function looks like a quadratic.

Taylor-expand $f(\mathbf{x})$ around the current solution $\mathbf{x}_k$ up to 2nd order:

$$f(\mathbf{x}_k + \boldsymbol{\varepsilon}) = f(\mathbf{x}_k) + \boldsymbol{\varepsilon}^T \mathbf{g}(\mathbf{x}_k) + \frac{1}{2} \boldsymbol{\varepsilon}^T H(\mathbf{x}_k) \boldsymbol{\varepsilon} + o\left(\boldsymbol{\varepsilon}^3\right)$$

where $\mathbf{g}(\mathbf{x}_k)$ and $H(\mathbf{x}_k)$ are the gradient and Hessian of f at $\mathbf{x}_k$:

$$\mathbf{g}(\mathbf{x}_k) \triangleq \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_k} \qquad H(\mathbf{x}_k) \triangleq \left. \frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{x}^T} \right|_{\mathbf{x}=\mathbf{x}_k}$$

Assuming H is (symmetric) positive definite, the next solution is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \arg\min_{\boldsymbol{\varepsilon}} \left\{ \boldsymbol{\varepsilon}^T \mathbf{g} + \frac{1}{2} \boldsymbol{\varepsilon}^T H \boldsymbol{\varepsilon} \right\} = \mathbf{x}_k - H^{-1} \mathbf{g}$$

Stabilizing Newton's method

For *convex* functions the Hessian H is always s.p.d, so the above method converges (usually quickly) to the global minimum. In reality however most functions we want to optimize are non-convex, which causes two problems:

1. H may be singular, which means that $\mathbf{x}_{k+1} = \mathbf{x}_k - H^{-1}\mathbf{g}$ will take us all the way to infinity.
2. H may have negative eigenvalues, which measn that (even if $\mathbf{x}_{k+1}$ is finite) we end up finding saddle points – minimum in some directions, maximum in other directions.

Stabilizing Newton's method

For *convex* functions the Hessian H is always s.p.d, so the above method converges (usually quickly) to the global minimum. In reality however most functions we want to optimize are non-convex, which causes two problems:

1. H may be singular, which means that $\mathbf{x}_{k+1} = \mathbf{x}_k - H^{-1}\mathbf{g}$ will take us all the way to infinity.
2. H may have negative eigenvalues, which measn that (even if $\mathbf{x}_{k+1}$ is finite) we end up finding saddle points – minimum in some directions, maximum in other directions.

These problems can be avoided in two general ways:

1. Trust region: minimize $\boldsymbol{\varepsilon}^{\mathsf{T}}\mathbf{g} + \frac{1}{2}\boldsymbol{\varepsilon}^{\mathsf{T}}H\boldsymbol{\varepsilon}$ s.t. $\|\boldsymbol{\varepsilon}\| \leq r$, where r is adapted over iterations. The minimization is usually done approximately.
2. Convexification/linearsearch: replace H with $H + \lambda I$, and/or use backtracking linesearch starting at the Newton point. When λ is large, $\mathbf{x}_k - (H + \lambda I)^{-1}\mathbf{g} \approx \mathbf{x}_k - \lambda^{-1}\mathbf{g}$, which is gradient descent with step λ^{-1}. The Levenberg-Marquardt method adapts λ over iterations.

Relation to linear solvers

The quadratic function

$$f(\mathbf{x}_k + \boldsymbol{\varepsilon}) = f(\mathbf{x}_k) + \boldsymbol{\varepsilon}^{\mathsf{T}} \mathbf{g}(\mathbf{x}_k) + \frac{1}{2} \boldsymbol{\varepsilon}^{\mathsf{T}} H(\mathbf{x}_k) \boldsymbol{\varepsilon}$$

is minimized when the gradient w.r.t $\boldsymbol{\varepsilon}$ vanishes, i.e. when

$$H\boldsymbol{\varepsilon} = -\mathbf{g}$$

When H is s.p.d, one can use the conjugate-gradient method for solving linear equations to do numerical optimization.

The set of vectors $\{\mathbf{v}_k\}_{k=1\cdots n}$ are conjugate if they satisfy $\mathbf{v}_i^{\mathsf{T}} H \mathbf{v}_j = 0$ for $i \neq j$. These are good search directions because they yield exact minimization of an n-dimensional quadratic in n iterations (using exact linesearch). Such a set can be constructed using Lanczos iteration:

$$s_{k+1}\mathbf{v}_{k+1} = (H - \alpha_k I)\mathbf{v}_k - s_k \mathbf{v}_{k-1}$$

where s_{k+1} is such that $\|\mathbf{v}_{k+1}\| = 1$, and $\alpha_k = \mathbf{v}_k^{\mathsf{T}} H \mathbf{v}_k$. Note that access to H is not required; all we need to be able to compute is $H\mathbf{v}$.

Non-linear least squares

Many optimization problems are in the form

$$f(\mathbf{x}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|^2$$

where $\mathbf{r}(\mathbf{x})$ is a vector of "residuals". Define the Jacobian of the residuals:

$$J(\mathbf{x}) = \frac{\partial \mathbf{r}(\mathbf{x})}{\partial \mathbf{x}}$$

Then the gradient and Hessian of f are

$$\begin{aligned} \mathbf{g}(\mathbf{x}) &= J(\mathbf{x})^\mathsf{T} \mathbf{r}(\mathbf{x}) \\ H(\mathbf{x}) &= J(\mathbf{x})^\mathsf{T} J(\mathbf{x}) + \frac{\partial J(\mathbf{x})}{\partial \mathbf{x}} \times \mathbf{r}(\mathbf{x}) \end{aligned}$$

We can omit the last term and obtain the Gauss-Netwon approximation:

$$H(\mathbf{x}) \approx J(\mathbf{x})^\mathsf{T} J(\mathbf{x})$$

Then Newton's method (with stabilization) becomes

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left(J_k^\mathsf{T} J_k + \lambda_k I\right)^{-1} J_k^\mathsf{T} \mathbf{r}_k$$