24. NLP algorithms

- Overview
- Local methods
- Constrained optimization
- Global methods
- Black-box methods
- Course wrap-up

Laurent Lessard (www.laurentlessard.com)

Review of algorithms

Studying **Linear Programs**, we talked about:

- **Simplex method:** traverse the *surface* of the feasible polyhedron looking for the vertex with minimum cost. Only applicable for linear programs. Used by solvers such as `Clp` and `CPLEX`. Hybrid versions used by `Gurobi` and `Mosek`.

- **Interior point methods:** traverse the *inside* of the feasible polyhedron and move towards the boundary point with minimum cost. Applicable to many different types of optimization problems. Used by `SCS`, `ECOS`, `Ipopt`.

Review of algorithms

Studying **Mixed Integer Programs**, we talked about:

- **Cutting plane methods:** solve a sequence of LP relaxations and keep adding cuts (special extra linear constraints) until solution is integral, and therefore optimal. Also applicable for more general convex problems.
- **Branch and bound methods:** solve a sequence of LP relaxations (upper bounding), and branch on fractional variables (lower bounding). Store problems in a tree, prune branches that aren't fruitful. Most optimization problems can be solved this way. You just need a way to *branch* (split the feasible set) and a way to *bound* (efficiently relax).
- Variants of methods above are used by all MIP solvers.

Overview of NLP algorithms

To solve **Nonlinear Programs** with **continuous variables**, there is a wide variety of available algorithms. We'll assume the problem has the standard form:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f_0(x) \\ \text{subject to:} & f_i(x) \le 0 \quad \text{for } i = 1, \dots, m \end{array}$$

- What works best depends on the kind of problem you're solving. We need to talk about problem categories.

Overview of NLP algorithms

1. Are the functions differentiable? Can we efficiently compute gradients or second derivatives of the f_i?
2. What problem size are we dealing with? a few variables and constraints? hundreds? thousands? millions?
3. Do we want to find local optima, or do we need the global optimum (more difficult!)
4. Does the objective function have a large number of local minima? or a relatively small number?

Overview of NLP algorithms

1. Are the functions differentiable? Can we efficiently compute gradients or second derivatives of the f_i?
2. What problem size are we dealing with? a few variables and constraints? hundreds? thousands? millions?
3. Do we want to find local optima, or do we need the global optimum (more difficult!)
4. Does the objective function have a large number of local minima? or a relatively small number?

Note: items **3** and **4** don't matter if the problem is convex. In that case any local minimum is also a global minimum!

Survey of NLP algorithms

- Local methods using derivative information. It's what most NLP solvers use (and what most JuMP solvers use).
  - unconstrained case
  - constrained case
- Global methods
- Derivative-free methods

Local methods using derivatives

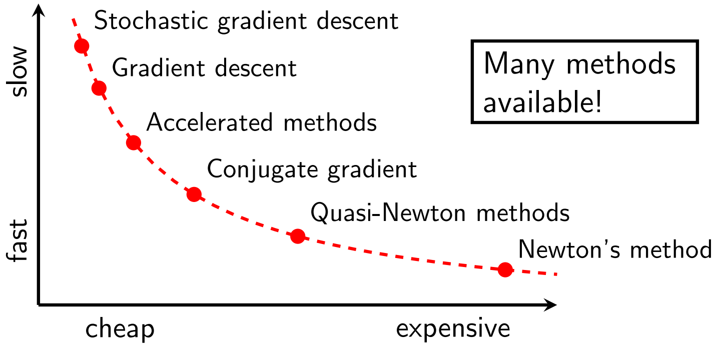
Let's start with the unconstrained case:

$$\underset{x}{\text{minimize}} \quad f(x)$$

Local methods using derivatives

Let's start with the unconstrained case:

$$\underset{x}{\text{minimize}} \quad f(x)$$



Many methods available!

Iterative methods

Local methods iteratively step through the space looking for a point where $\nabla f(x) = 0$.

1. pick a starting point x_0.
2. choose a direction to move in Δ_k. This is the part where different algorithms do different things.
3. update your location $x_{k+1} = x_k + \Delta_k$
4. repeat until you're happy with the function value or the algorithm has ceased to make progress.

Vector calculus

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a twice-differentiable function.

- The **gradient** of f is a function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ defined by:

$$\left[\nabla f\right]_i = \frac{\partial f}{\partial x_i}$$

  $\nabla f(x)$ points in the direction of *greatest increase* of f at x.

Vector calculus

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a twice-differentiable function.

- The **gradient** of f is a function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ defined by:

$$\left[\nabla f\right]_i = \frac{\partial f}{\partial x_i}$$

  $\nabla f(x)$ points in the direction of *greatest increase* of f at x.

- The **Hessian** of f is a function $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ where:

$$\left[\nabla^2 f\right]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

  $\nabla^2 f(x)$ is a matrix that encodes the *curvature* of f at x.

Vector calculus

Example: suppose $f(x, y) = x^2 + 3xy + 5y^2 - 7x + 2$

- $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$

Vector calculus

Example: suppose $f(x, y) = x^2 + 3xy + 5y^2 - 7x + 2$

- $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + 3y - 7 \\ 3x + 10y \end{bmatrix}$

- $\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$

Vector calculus

Example: suppose $f(x, y) = x^2 + 3xy + 5y^2 - 7x + 2$

- $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + 3y - 7 \\ 3x + 10y \end{bmatrix}$

- $\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 3 & 10 \end{bmatrix}$

Vector calculus

Example: suppose $f(x,y) = x^2 + 3xy + 5y^2 - 7x + 2$

- $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + 3y - 7 \\ 3x + 10y \end{bmatrix}$
- $\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 3 & 10 \end{bmatrix}$

Taylor's theorem in n dimensions

$$f(x) \approx f(x_0) + \nabla f(x_0)^{\mathsf{T}}(x - x_0) + \frac{1}{2}(x - x_0)^{\mathsf{T}} \nabla^2 f(x_0)(x - x_0) + \dots$$

Vector calculus

Example: suppose $f(x,y) = x^2 + 3xy + 5y^2 - 7x + 2$

- $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + 3y - 7 \\ 3x + 10y \end{bmatrix}$
- $\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 3 & 10 \end{bmatrix}$

Taylor's theorem in n dimensions

$$f(x) \approx \underbrace{\overbrace{f(x_0) + \nabla f(x_0)^\mathsf{T}(x - x_0)}^{\text{best linear approximation}} + \frac{1}{2}(x - x_0)^\mathsf{T}\nabla^2 f(x_0)(x - x_0)}_{\text{best quadratic approximation}} + \dots$$

Gradient descent

- The simplest of all iterative methods. It's a first-order method, which means it only uses gradient information:

$$x_{k+1} = x_k - t_k \nabla f(x_k)$$

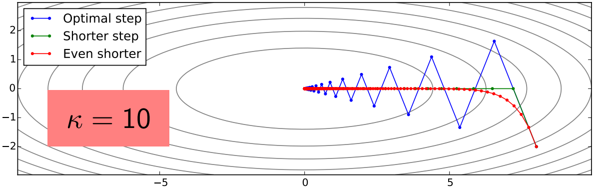
- $-\nabla f(x_k)$ points in the direction of local steepest decrease of the function. We will move in this direction.
- t_k is the stepsize. Many ways to choose it:
  - Pick a constant $t_k = t$
  - Pick a slowly decreasing stepsize, such as $t_k = 1/\sqrt{k}$
  - Exact line search: $t_k = \arg\min_t f(x_k - t\nabla f(x_k))$.
  - A heuristic method (most common in practice).
    Example: backtracking line search.

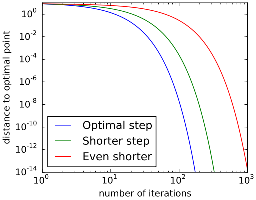
Gradient descent

We can gain insight into the effectiveness of a method by seeing how it performs on a quadratic: $f(x) = \frac{1}{2}x^\mathsf{T}Qx$. The **condition number** $\kappa := \frac{\lambda_{\mathsf{max}}(Q)}{\lambda_{\mathsf{min}}(Q)}$ determines convergence.

Gradient descent

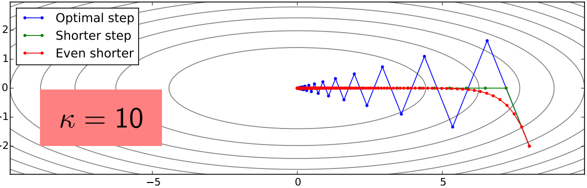
We can gain insight into the effectiveness of a method by seeing how it performs on a quadratic: $f(x) = \frac{1}{2}x^\mathsf{T}Qx$. The **condition number** $\kappa := \frac{\lambda_{\max}(Q)}{\lambda_{\min}(Q)}$ determines convergence.

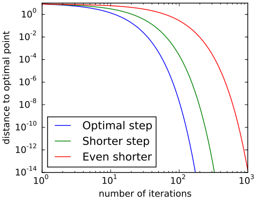


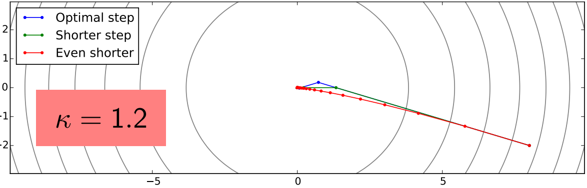


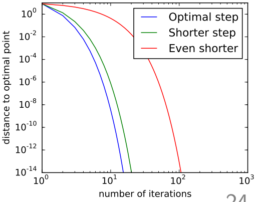
Gradient descent

We can gain insight into the effectiveness of a method by seeing how it performs on a quadratic: $f(x) = \frac{1}{2}x^\mathsf{T}Qx$. The **condition number** $\kappa := \frac{\lambda_{\max}(Q)}{\lambda_{\min}(Q)}$ determines convergence.









Gradient descent

Advantages

- Simple to implement and cheap to execute.
- Can be easily adjusted.
- Robust in the presence of noise and uncertainty.

Gradient descent

Advantages

- Simple to implement and cheap to execute.
- Can be easily adjusted.
- Robust in the presence of noise and uncertainty.

Disadvantages

- Convergence is slow.
- Sensitive to conditioning. Even rescaling a variable can have a substantial effect on performance!
- Not always easy to tune the stepsize.

Gradient descent

Advantages

- Simple to implement and cheap to execute.
- Can be easily adjusted.
- Robust in the presence of noise and uncertainty.

Disadvantages

- Convergence is slow.
- Sensitive to conditioning. Even rescaling a variable can have a substantial effect on performance!
- Not always easy to tune the stepsize.

Note: The idea of preconditioning (rescaling) before solving adds another layer of possible customizations and tradeoffs.

Other first-order methods

Accelerated methods

- Still a first-order method, but makes use of past iterates to accelerate convergence. Example: the Heavy-ball method:

  $$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1})$$

  Other examples: Nesterov, Beck & Teboulle, others.

Other first-order methods

Accelerated methods

- Still a first-order method, but makes use of past iterates to accelerate convergence. Example: the Heavy-ball method:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1})$$

  Other examples: Nesterov, Beck & Teboulle, others.
- Can achieve substantial improvement over gradient descent with only a moderate increase in computational cost
- Not as robust to noise as gradient descent, and can be more difficult to tune because there are more parameters.

Other first-order methods

Stochastic gradient descent

- Similar to gradient descent, but only evaluate some of the components of $\nabla f(x_k)$, chosen at random.
- Same pros and cons as gradient descent, but allows further tradeoff of speed vs computation.
- Industry standard for big-data problems like deep learning.

Other first-order methods

Stochastic gradient descent

- Similar to gradient descent, but only evaluate some of the components of $\nabla f(x_k)$, chosen at random.
- Same pros and cons as gradient descent, but allows further tradeoff of speed vs computation.
- Industry standard for big-data problems like deep learning.

Nonlinear conjugate gradient

- Variant of the standard conjugate gradient algorithm for solving $Ax = b$, but adapted for use in general optimization.
- Requires more computation than accelerated methods.
- Converges exactly in a finite number of steps when applied to quadratic functions.

Newton's method

Basic idea: approximate the function as a quadratic, move directly to the minimum of that quadratic, and repeat.

Newton's method

Basic idea: approximate the function as a quadratic, move directly to the minimum of that quadratic, and repeat.

- If we're at x_k, then by Taylor's theorem:

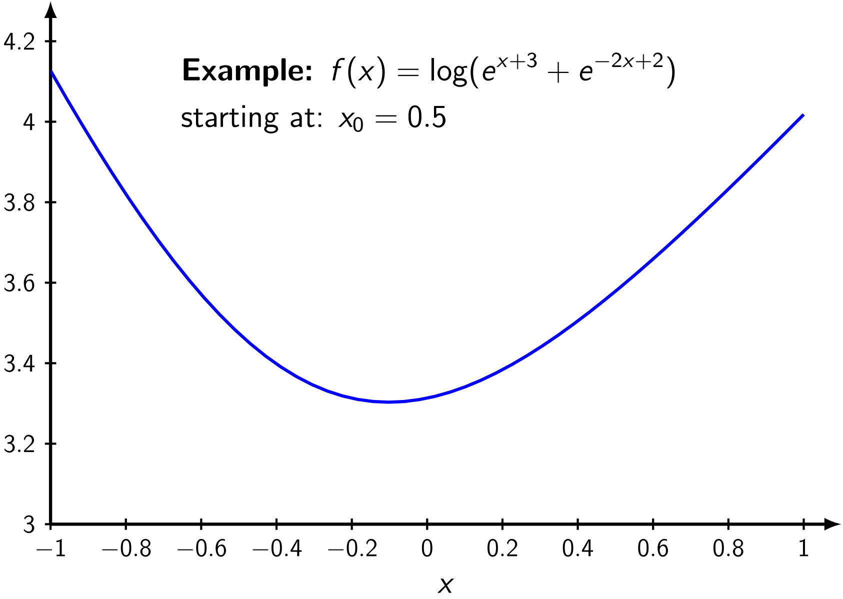
$$f(x) \approx f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_0) + \frac{1}{2}(x - x_k)^\mathsf{T} \nabla^2 f(x_k)(x - x_k)$$

- If $\nabla^2 f(x_k) \succ 0$, the minimum of the quadratic occurs at:

$$x_{k+1} := x_{\text{opt}} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k)$$

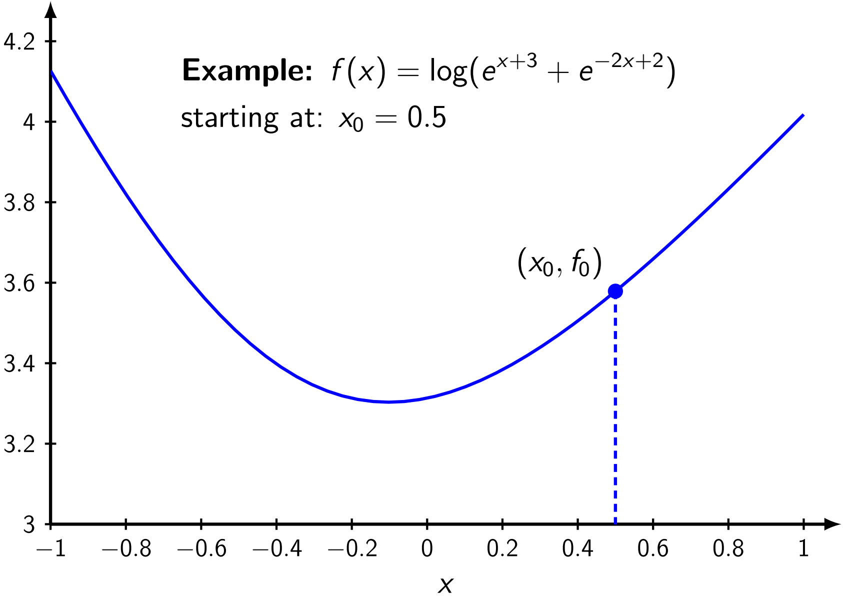
- Newton's method is a second-order method; it requires computing the Hessian (second derivatives).

Newton's method in 1D



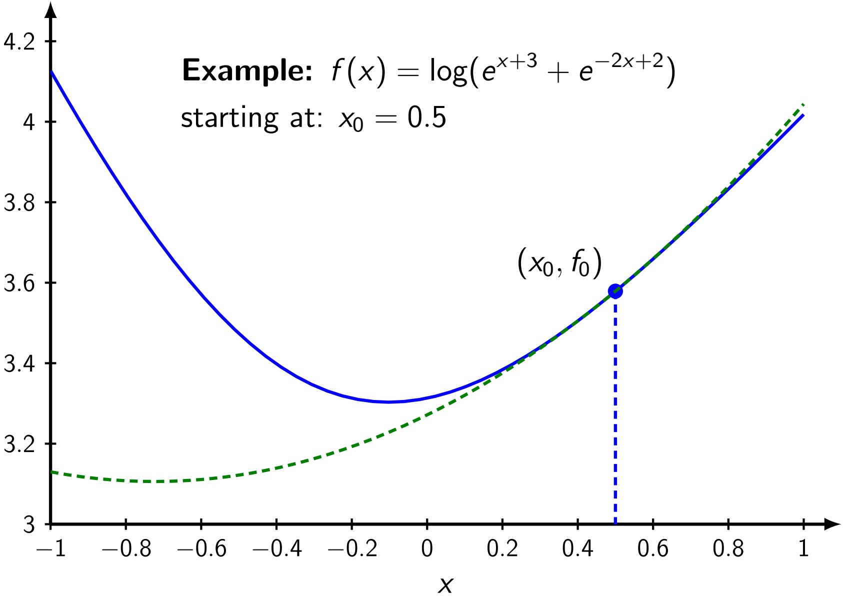
example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D



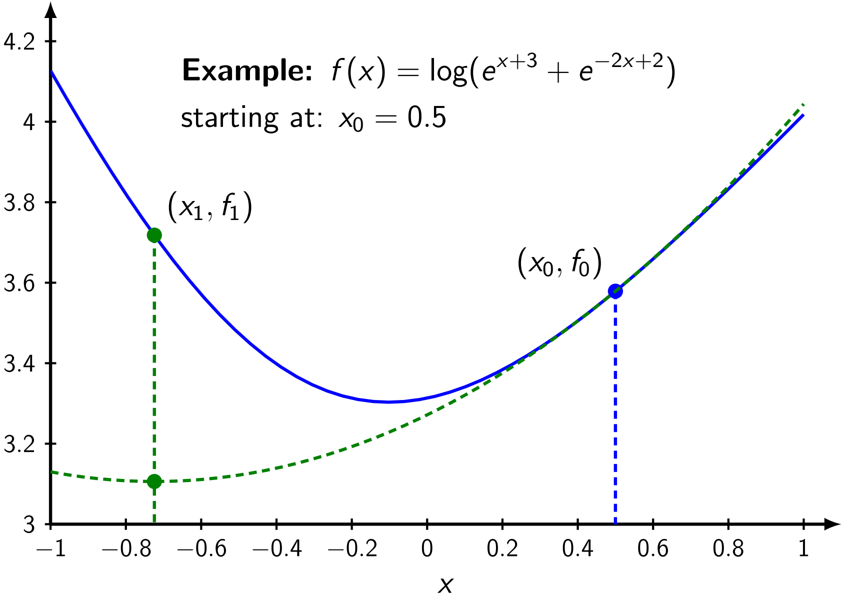
example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D



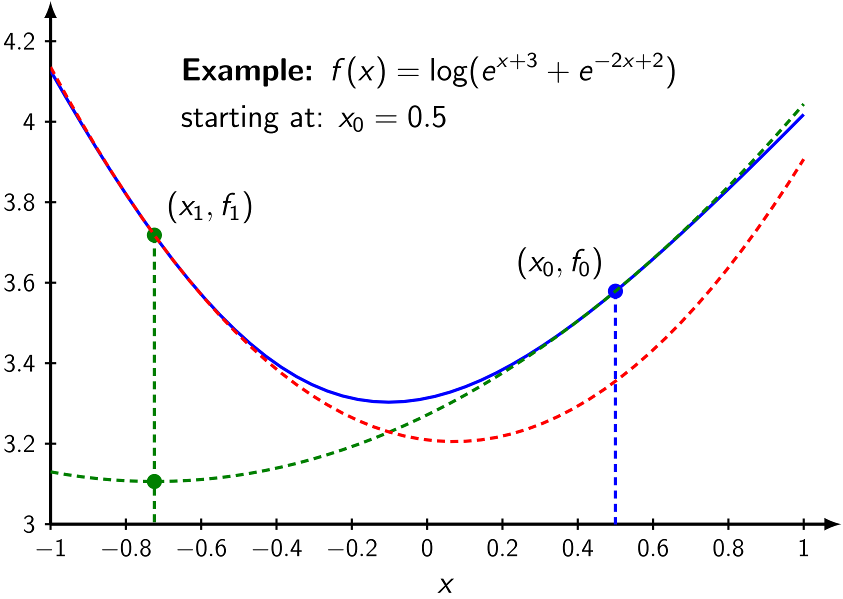
example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D



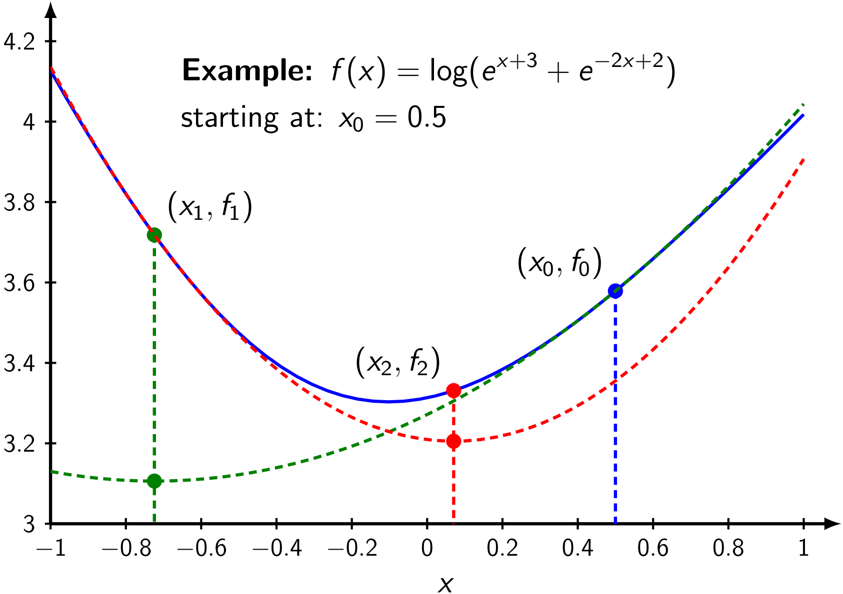
example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D



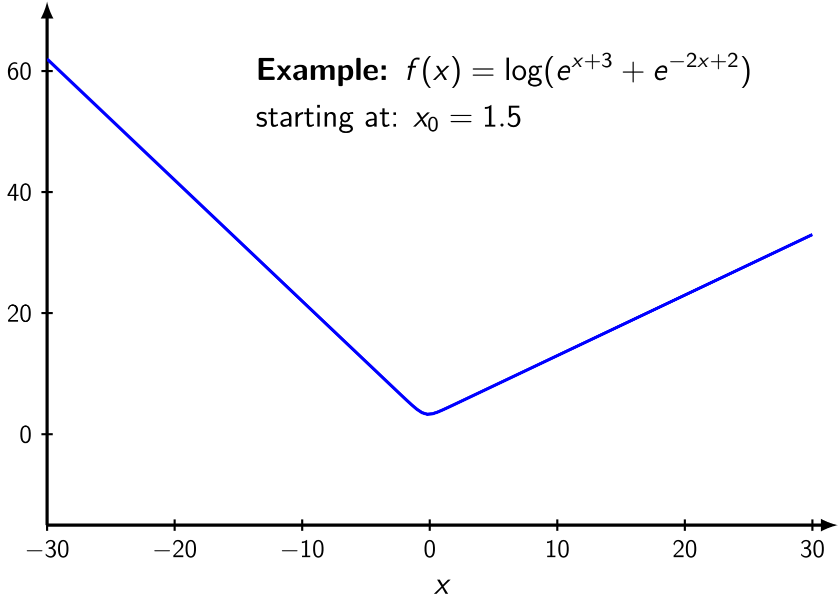
example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D



example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D

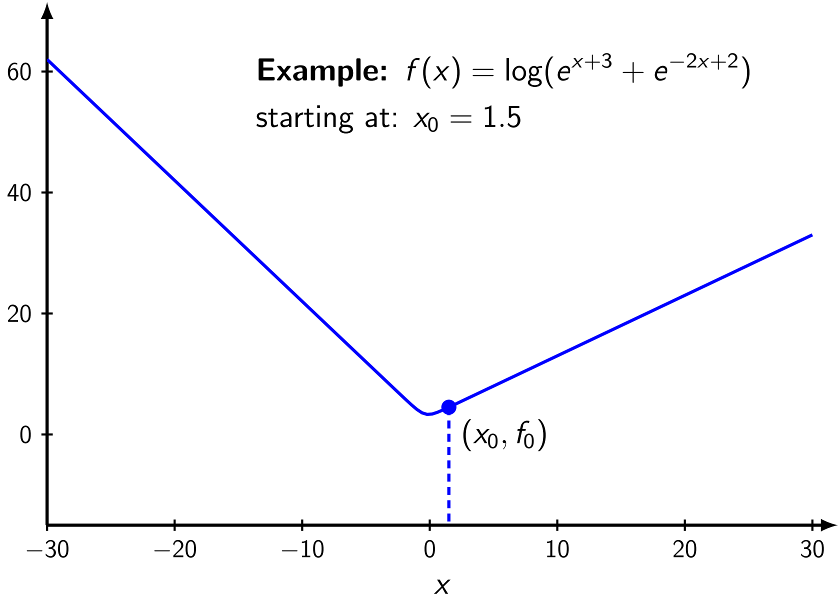


Example: $f(x) = \log(e^{x+3} + e^{-2x+2})$

starting at: $x_0 = 1.5$

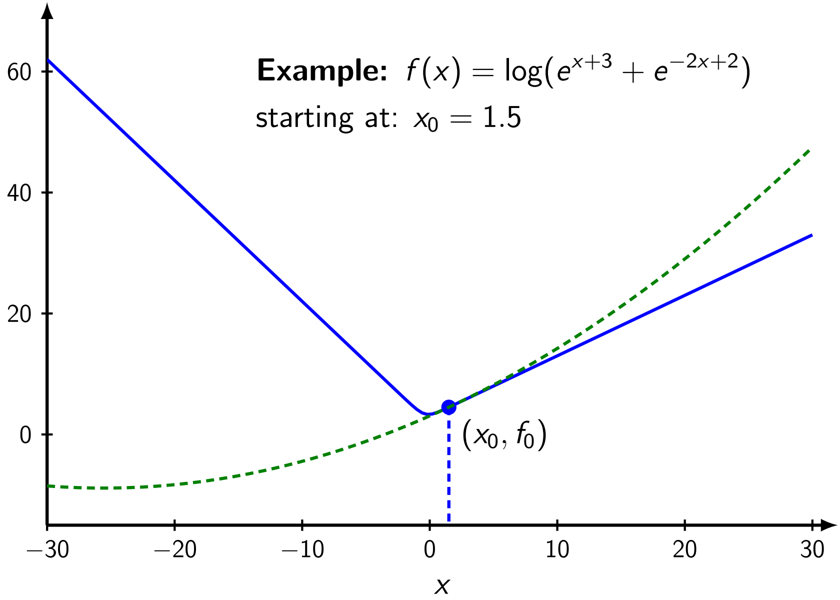
example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D



example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D

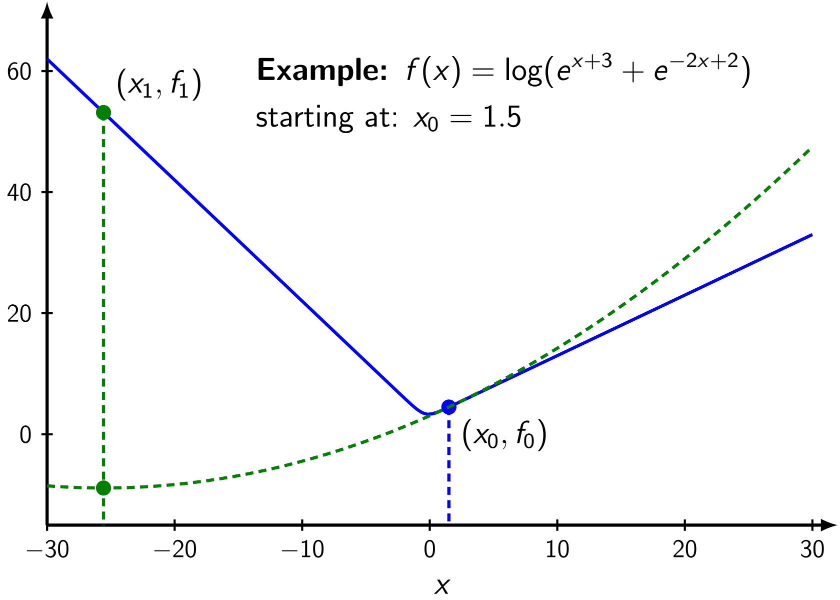


example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D

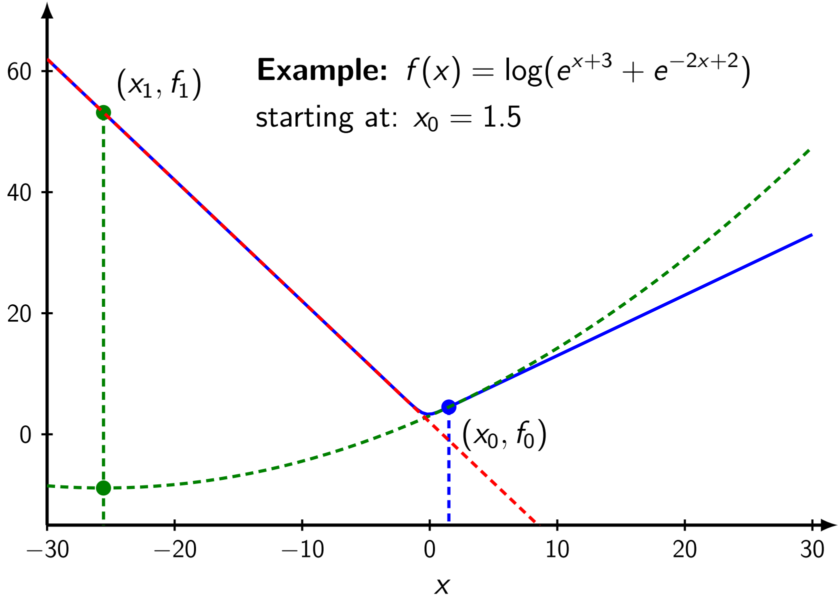
Example: $f(x) = \log(e^{x+3} + e^{-2x+2})$

starting at: $x_0 = 1.5$



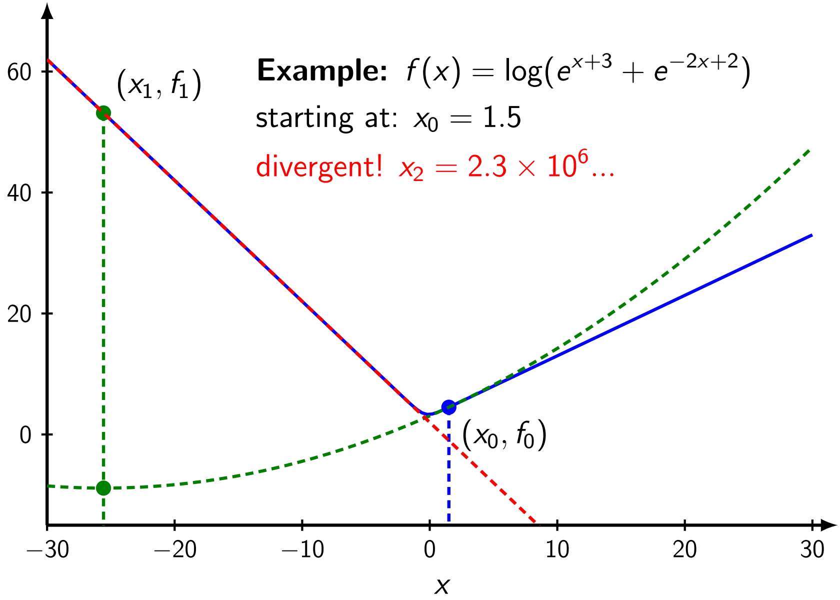
example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D



example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method in 1D



example by: L. El Ghaoui, UC Berkeley, EE127a

Newton's method

Advantages

- It's usually *very* fast. Converges to the exact optimum in one iteration if the objective is quadratic.
- It's scale-invariant. Convergence rate is not affected by any linear scaling or transformation of the variables.

Newton's method

Advantages

- It's usually *very* fast. Converges to the exact optimum in one iteration if the objective is quadratic.
- It's scale-invariant. Convergence rate is not affected by any linear scaling or transformation of the variables.

Disadvantages

- If n is large, storing the Hessian (an $n \times n$ matrix) and computing $\nabla^2 f(x_k)^{-1} \nabla f(x_k)$ can be prohibitively expensive.
- If $\nabla^2 f(x_k) \nsucc 0$, Newton's method may converge to a local maximum or a saddle point.
- May fail to converge at all if we start too far from the optimal point.

Quasi-Newton methods

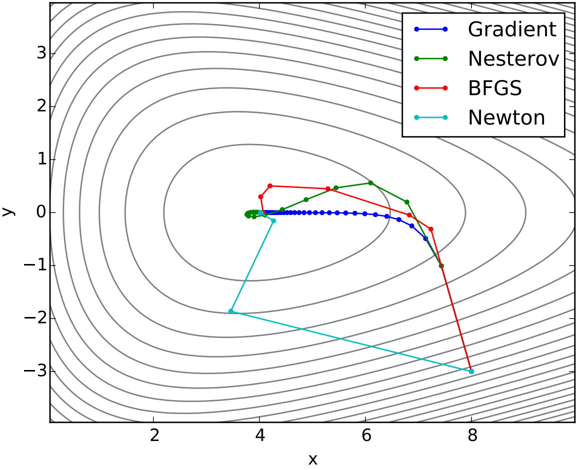
- An approximate Newton's method that doesn't require computing the Hessian.
- Uses an approximation $H_k \approx \nabla^2 f(x_k)^{-1}$ that can be updated directly and is faster to compute than the full Hessian.

$$\begin{aligned} x_{k+1} &= x_k - H_k \nabla f(x_k) \\ H_{k+1} &= g(H_k, \nabla f(x_k), x_k) \end{aligned}$$

- Several popular update schemes for H_k:
  - DFP (Davidon–Fletcher–Powell)
  - BFGS (Broyden–Fletcher–Goldfarb–Shanno)

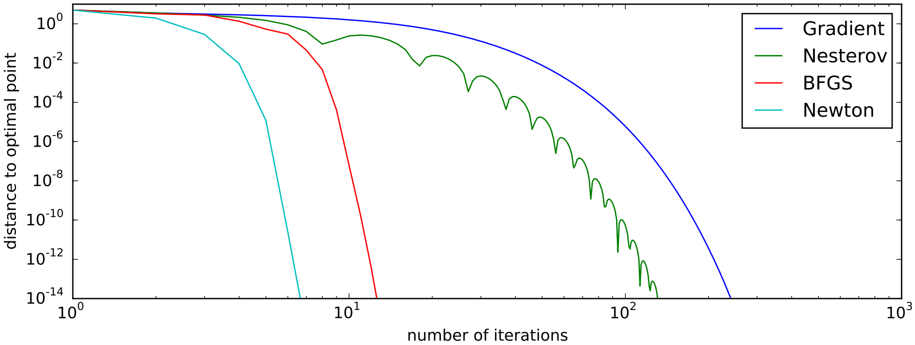
Example

- $f(x, y) = e^{-(x-3)/2} + e^{(x+4y)/10} + e^{(x-4y)/10}$
- Function is smooth, with a single minimum near $(4.03, 0)$.



Example

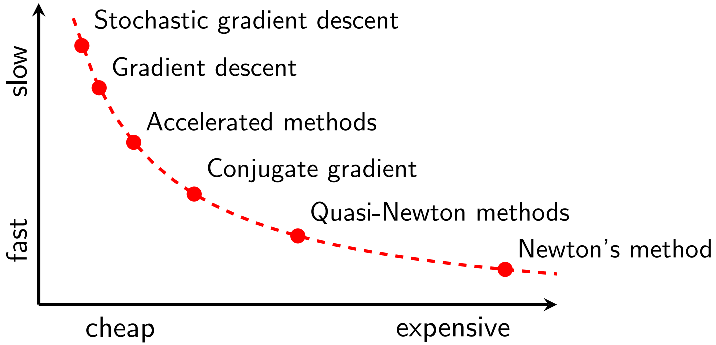
Plot showing iterations to convergence:



- Illustrates the complexity vs performance tradeoff.
- Nesterov's method doesn't always converge uniformly.
- Julia code: IterativeMethods.ipynb

Recap of local methods

Important: For any of the local methods we've seen, if $\nabla f(x_k) = 0$, then $x_{k+1} = x_k$ and we we won't move!



Constrained local optimization

Algorithms we've seen so far are designed for *unconstrained* optimization. How do we deal with constraints?

Constrained local optimization

Algorithms we've seen so far are designed for *unconstrained* optimization. How do we deal with constraints?

- We'll revisit **interior point methods**, and we'll also talk about a class of algorithms called **active set methods**.
- These are among the most popular methods for smooth constrained optimization.

Interior point methods

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to:} \quad & f_i(x) \leq 0 \end{aligned}$$

Interior point methods

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to:} \quad & f_i(x) \le 0 \end{aligned}$$

Basic idea: augment the objective function using a **barrier** that goes to infinity as we approach a constraint.

Interior point methods

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to:} \quad & f_i(x) \leq 0 \end{aligned}$$

Basic idea: augment the objective function using a **barrier** that goes to infinity as we approach a constraint.

$$\underset{x}{\text{minimize}} \quad f_0(x) - \mu \sum_{i=1}^{m} \log\big(-f_i(x)\big)$$

Interior point methods

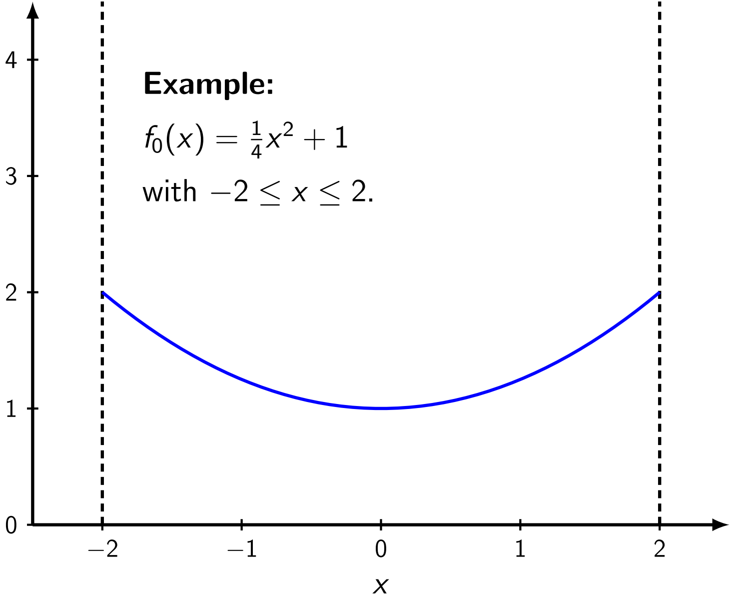
$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to:} \quad & f_i(x) \leq 0 \end{aligned}$$

Basic idea: augment the objective function using a **barrier** that goes to infinity as we approach a constraint.

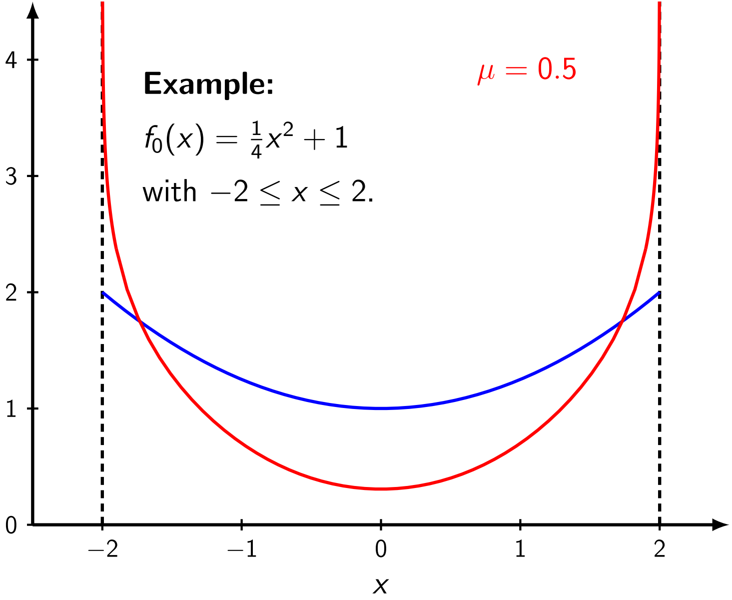
$$\underset{x}{\text{minimize}} \quad f_0(x) - \mu \sum_{i=1}^{m} \log\bigl(-f_i(x)\bigr)$$

Then, alternate between **(1)** an iteration of an unconstrained method (usually Newton's) and **(2)** shrinking μ toward zero.

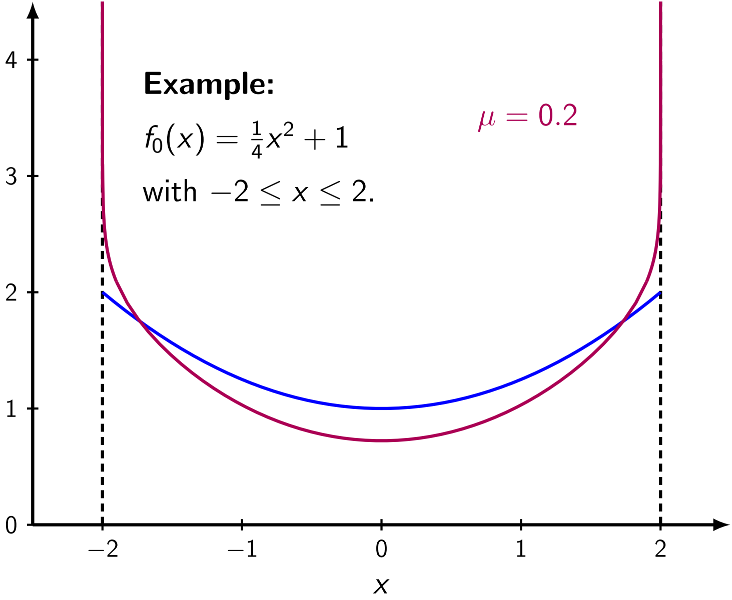
Interior point methods



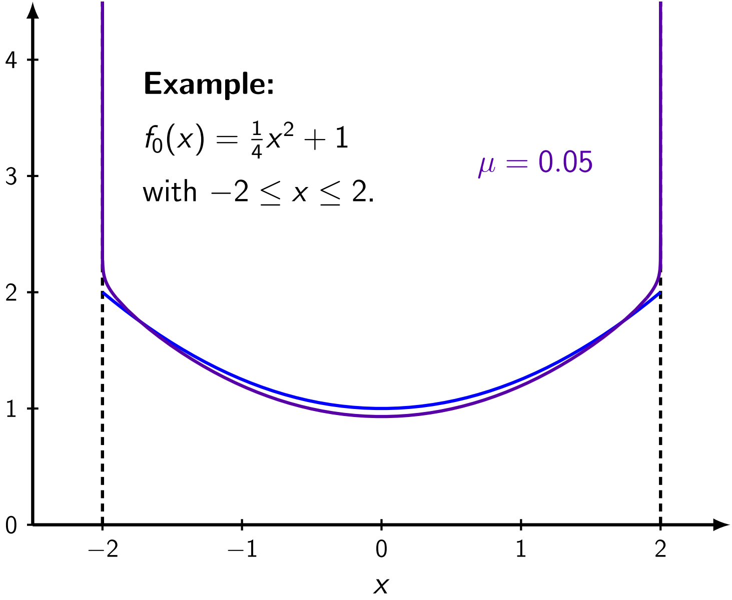
Interior point methods



Interior point methods



Interior point methods



Active set methods

$$\begin{array}{rl} \underset{x}{\text{minimize}} & f_0(x) \\ \text{subject to:} & f_i(x) \leq 0 \end{array}$$

Basic idea: at optimality, some of the constraints will be **active** (equal to zero). The others can be ignored.

Active set methods

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to:} \quad & f_i(x) \le 0 \end{aligned}$$

Basic idea: at optimality, some of the constraints will be **active** (equal to zero). The others can be ignored.

- given some active set, we can solve or approximate the solution of the simultaneous equalities (constraints not in the active set are ignored). Approximations typically use linear (LP) or quadratic (QP) functions.
- inequality constraints are then added or removed from the active set based on certain rules, then repeat.
- the simplex method is an example of an active set method.

NLP solvers in JuMP

- `Ipopt` (**I**nterior **P**oint **OPT**imizer) uses an interior point method to handle constraints. If second derivative information is available, it uses a sparse Newton iteration, otherwise it uses a BFGS or SR1 (another Quasi-Newton method).

NLP solvers in JuMP

- `Ipopt` (**I**nterior **P**oint **OPT**imizer) uses an interior point method to handle constraints. If second derivative information is available, it uses a sparse Newton iteration, otherwise it uses a BFGS or SR1 (another Quasi-Newton method).
- `Knitro` (**N**onlinear **I**nterior point **T**rust **R**egion **O**ptimization) implements four different algorithms. Two are interior point (one is algebraic, the other uses conjugate-gradient as the solver). The other two are active set (one uses sequential LP approximations, the other uses sequential QP approximations).

NLP solvers in JuMP

- `Ipopt` (**I**nterior **P**oint **OPT**imizer) uses an interior point method to handle constraints. If second derivative information is available, it uses a sparse Newton iteration, otherwise it uses a BFGS or SR1 (another Quasi-Newton method).

- `Knitro` (**N**onlinear **I**nterior point **T**rust **R**egion **O**ptimization) implements four different algorithms. Two are interior point (one is algebraic, the other uses conjugate-gradient as the solver). The other two are active set (one uses sequential LP approximations, the other uses sequential QP approximations).

- `NLopt` is an open-source platform that interfaces with many (currently 43) different solvers. Only a handful are currently available in JuMP, but some are global/derivative-free.

NLopt solvers

http://ab-initio.mit.edu/wiki/index.php/NLopt_Algorithms

LD_AUGLAG	LN_AUGLAG	GN_CRS2_LM
LD_AUGLAG_EQ	LN_AUGLAG_EQ	GN_DIRECT
LD_CCSAQ	LN_BOBYQA	GN_DIRECT_L
LD_LBFGS_NOCEDAL	LN_COBYLA	GN_DIRECT_L_RAND
LD_LBFGS	LN_NEWUOA	GN_DIRECT_NOSCAL
LD_MMA	LN_NEWUOA_BOUND	GN_DIRECT_L_NOSCAL
LD_SLSQP	LN_NELDERMEAD	GN_DIRECT_L_RAND_NOSCAL
LD_TNEWTON	LN_PRAXIS	GN_ESCH
LD_TNEWTON_RESTART	LN_SBPLX	GN_ISRES
LD_TNEWTON_PRECOND	GD_MLSL	GN_MLSL
LD_TNEWTON_PRECOND_RESTART	GD_MLSL_LDS	GN_MLSL_LDS
LD_VAR1	GD_STOGO	GN_ORIG_DIRECT
LD_VAR2	GD_STOGO_RAND	GN_ORIG_DIRECT_L

- L/G: local/global method
- D/N: derivative-based/derivative-free
- mostly implemented in C++, some work with Julia/JuMP

Global methods

A global method makes an effort to find a **global** optimum rather than just a local one.

- If gradients are available, the standard (and obvious) thing to do is **multistart** (also known as *random restarts*).
  - Randomly pepper the space with initial points.
  - Run your favorite local method starting from each point (these runs can be executed in parallel).
  - Compare the different local minima found.
- The number of restarts required depends on the size of the space and how many local minima it contains.

Global methods

A global method makes an effort to find a **global** optimum rather than just a local one.

- A more sophisticated approach:
  - Systematically partition the space using a branch-and-bound technique.
  - Search the smaller spaces using local gradient-based search.
- Knowledge of derivatives is required for both the bounding and local optimization steps.

Black-box methods

What if no derivative information is available and all we can do is compute $f(x)$? We must resort to black-box methods (also known as: derivative-free or direct search methods).

If f is smooth:

- Approximate the derivative numerically by using finite differences, and then use a standard gradient-based method.
- Use coordinate descent: pick one coordinate, perform a line search, then pick the next coordinate, and keep cycling.

Black-box methods

What if no derivative information is available and f is not smooth? (you're usually in trouble)

Pattern search: Search in a grid and refine the grid adaptively in areas where larger variations are observed.

Genetic algorithms: Randomized approach that simulates a *population* of candidate points and uses a combination of *mutation* and *crossover* at each iteration to generate new candidate points. The idea is to mimic natural selection.

Simulated annealing: Randomized approach using gradient descent that is perturbed in proportion to a *temperature* parameter. Simulation continues as the system is progressively *cooled*. The idea is to mimic physics / crystalization.

Optimization at UW–Madison

- Linear programming and related topics
  - CS 525: linear programming methods
  - CS 526: advanced linear programming
- Convex optimization and iterative algorithms
  - CS 726: nonlinear optimization I
  - CS 727: nonlinear optimization II
  - CS 727: convex analysis
- MIP and combinatorial optimization
  - CS 425: introduction to combinatorial optimization
  - CS 577: introduction to algorithms
  - CS 720: integer programming
  - CS 728: integer optimization

Optimization at UW–Madison

- Plenty of applied courses as well:
  - machine learning
  - operations research
  - signal processing
  - robotics
  - ...

Optimization at UW–Madison

- Plenty of applied courses as well:
  - machine learning
  - operations research
  - signal processing
  - robotics
  - ...
- **ECE/CS/ME 532** (Fall 2017–18)
  Matrix Methods in Machine Learning.
  An introduction to machine learning from an applied linear algebra and optimization viewpoint. This class will make you understand linear algebra.

External resources

Continuous optimization

- Lieven Vandenberghe (UCLA) http://www.seas.ucla.edu/~vandenbe/
- Stephen Boyd (Stanford) http://web.stanford.edu/~boyd/
- Ryan Tibshirani (CMU) http://stat.cmu.edu/~ryantibs/convexopt/
- L. El Ghaoui (Berkeley) http://www.eecs.berkeley.edu/~elghaoui/

Discrete optimization

- Dimitris Bertsimas (MIT) – integer programming
  http://ocw.mit.edu/courses/sloan-school-of-management/15-083j-integer-programming-and-combinatorial-optimization-fall-2009/
- AM121 (Harvard) – intro to optimization
  http://am121.seas.harvard.edu/

Next week

- Project!

Thanks!