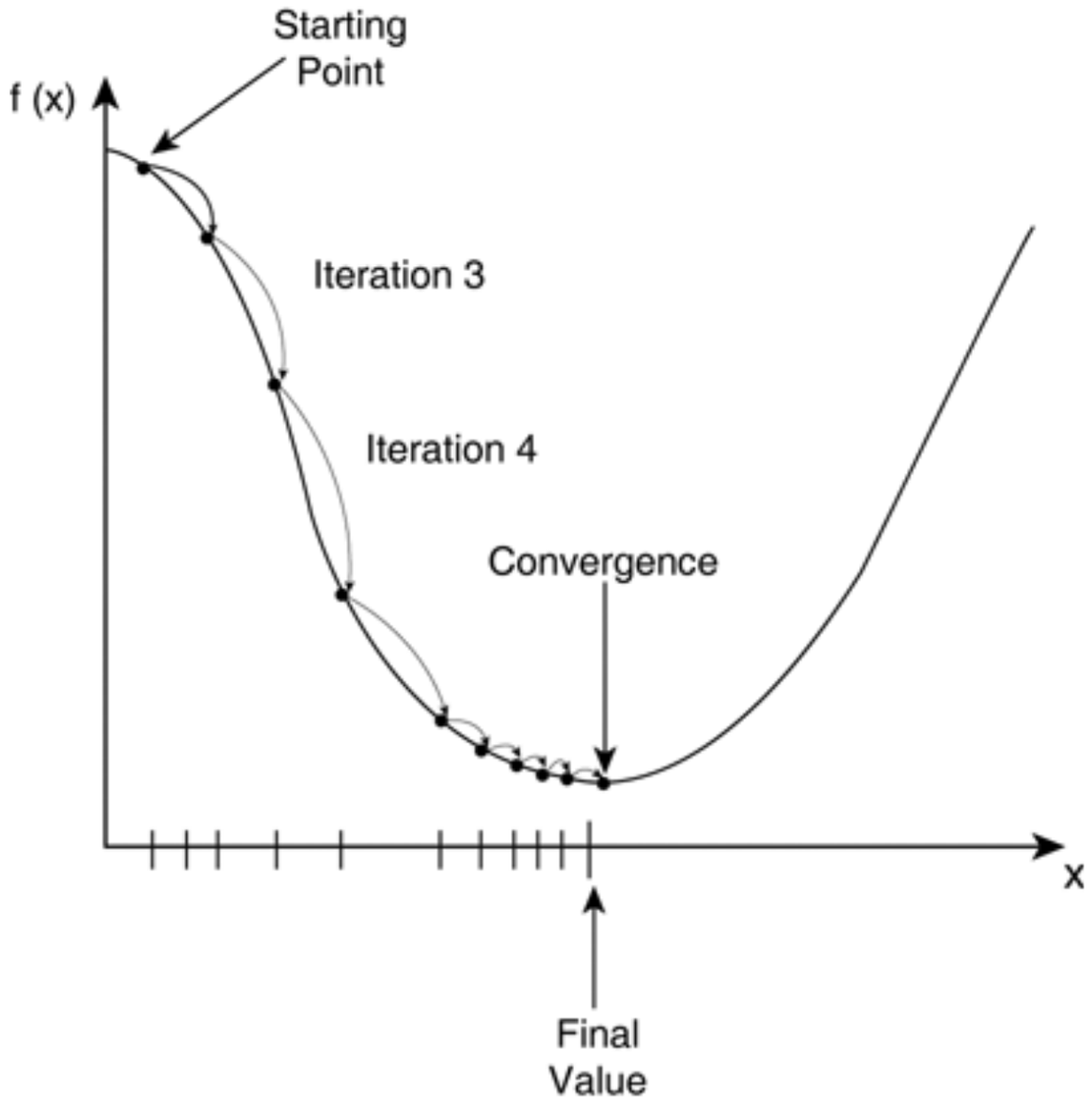
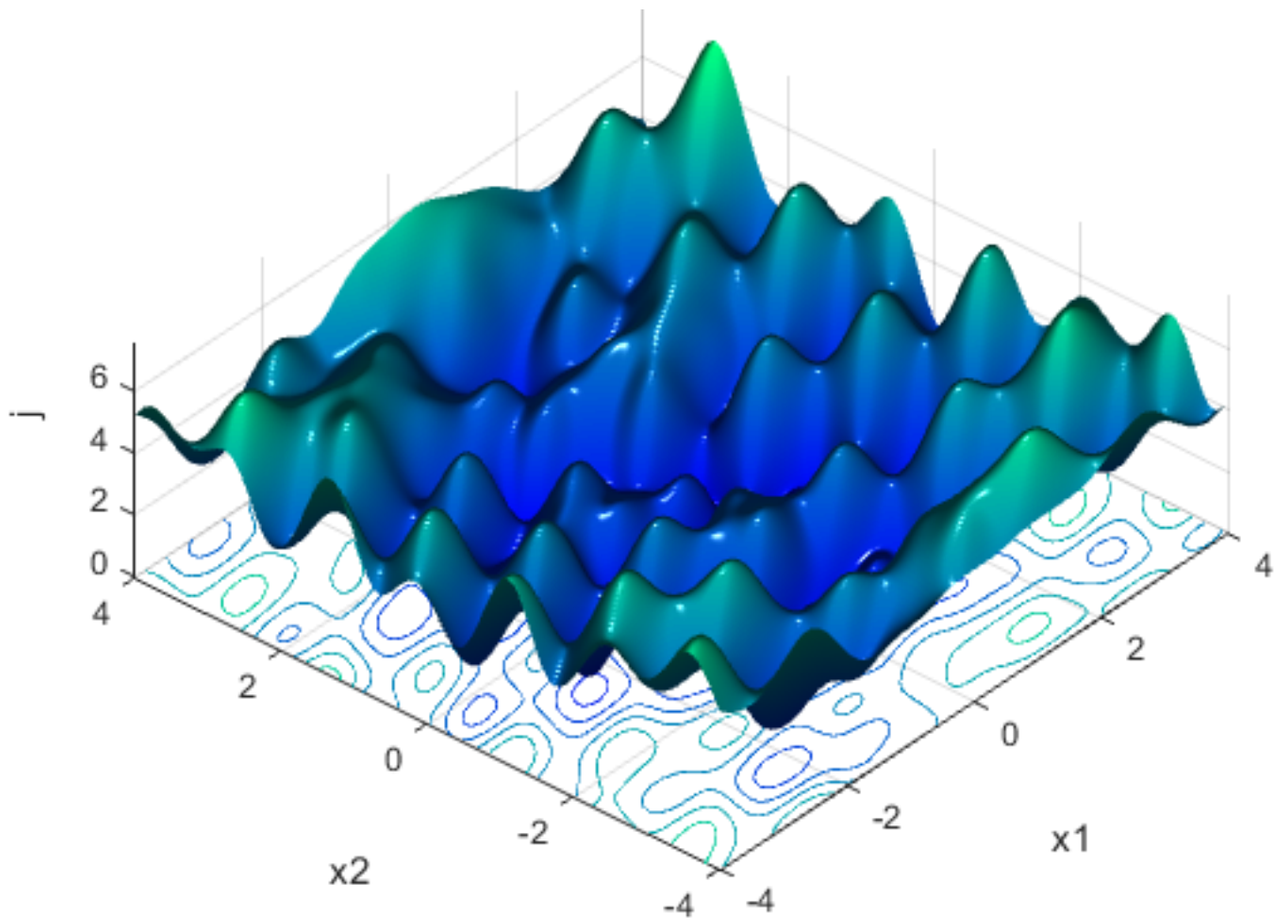
Optimization





Why should you care about the solution strategies?

- Understanding the optimization approaches behind the algorithms makes you more effectively choose which algorithm to run

- Understanding the optimization approaches makes you formalize your problem more effectively
  - otherwise you might formalize a very hard optimization problem; sometimes with minor modifications, can significantly simplify for the solvers, without impacting properties of solution significantly

- When you want to do something outside the given packages or solvers (which is often true)

- …also its fun!

Thought questions

- Many questions about existence and finding optimal solution
  - e.g., “…What if the maximum likelihood estimation of a parameter does not exist?…”
  - e.g., “…Do we always assume convex objectives?…”
  - e.g., How can we find the global solution, and not get stuck in local minima or saddlepoints?
  - e.g., Are local minima good enough?
  - e.g., How do we pick starting points?

Optimality

- We will not only deal with convex functions
  - We just have so far, and if we *can* make our optimization convex, then this is better
  - i.e., if you have two options (convex and non-convex), and its not clear one is better than the other, may as well pick the convex one
- The field of optimization deals with finding optimal solutions for non-convex problems
  - Sometimes possible, sometimes not possible
  - One strategy: random restarts
  - Another strategy: smart initialization approaches
- How do we pick a good starting point for gradient descent?
- Is a local minimum good enough?

How do we pick model types? Such as distributions and priors?

- For most ML problems, we will pick generic distributions that match the type of the target

- Where do priors come from? General purpose (e.g., regularizers, sparsity) or specified by an expert
  - e.g., imagine modelling the distribution over images of trees, with feature vector containing height, leave size, age, etc. An expert might know ranges and general relationships on these variables, to narrow the choice of distributions

- Suggested in TQs: Use some data to estimate a prior. Then, use that prior with new data. Would this be better than simply doing maximum likelihood to start?

Where does gradient descent come from?

- Goal is to find a stationary point, but cannot get closed form solution for gradient = 0

- Taylor series expansion with
  - First order for gradient descent
  - Second order for Newton-Raphson method (also called second-order gradient descent)

Taylor series expansion

A function $f(x)$ in the neighborhood of point x_0, can be approximated using the Taylor series as

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n,$$

where $f^{(n)}(x_0)$ is the n-th derivative of function $f(x)$ evaluated at point x_0.

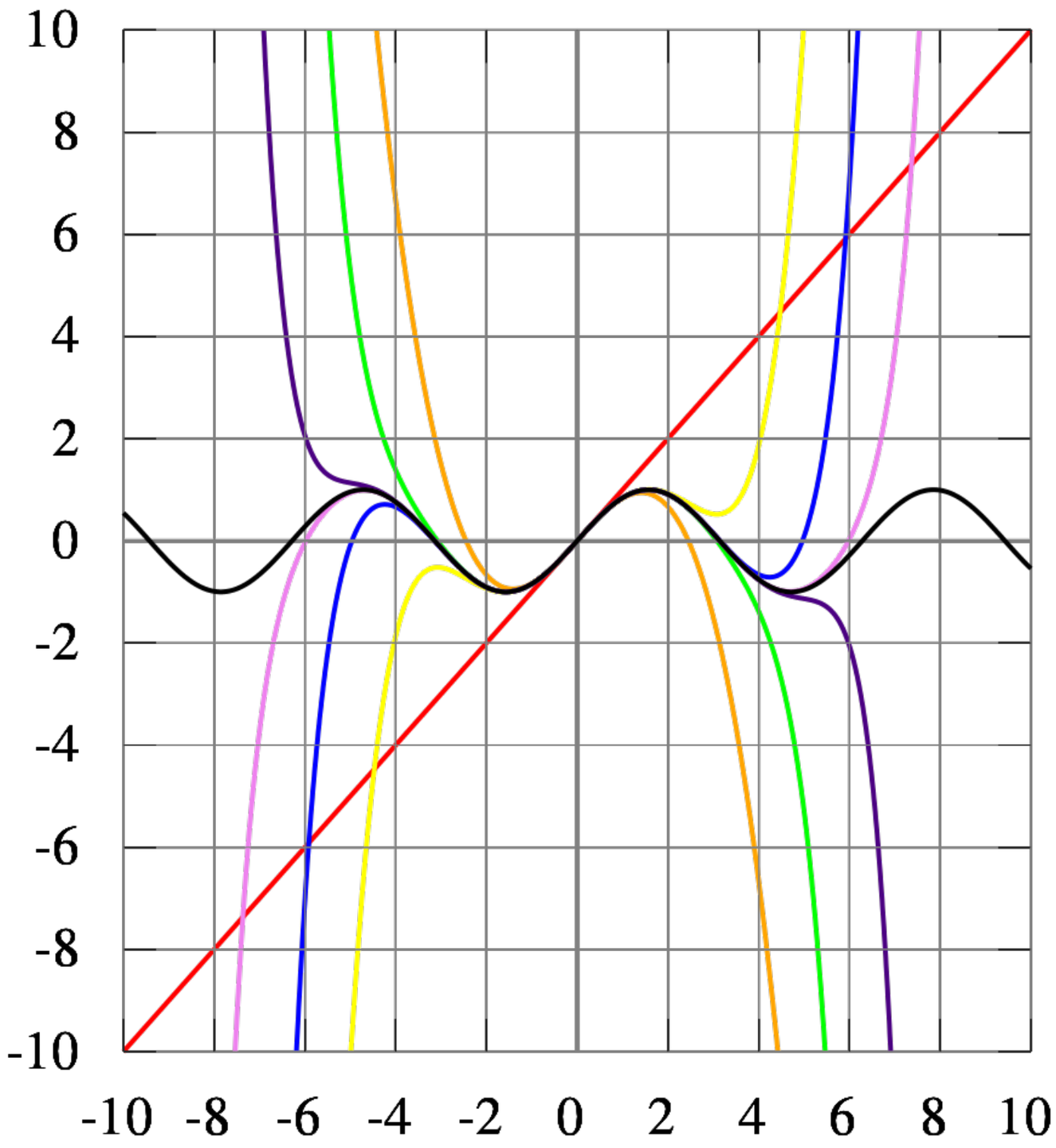
e.g. $$f(x) \approx f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0).$$

Taylor series expansion

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

Function: sin x
x0 = 0
Local approximation

degree **1**, **3**, **5**, **7**, **9**, **11** and **13**.

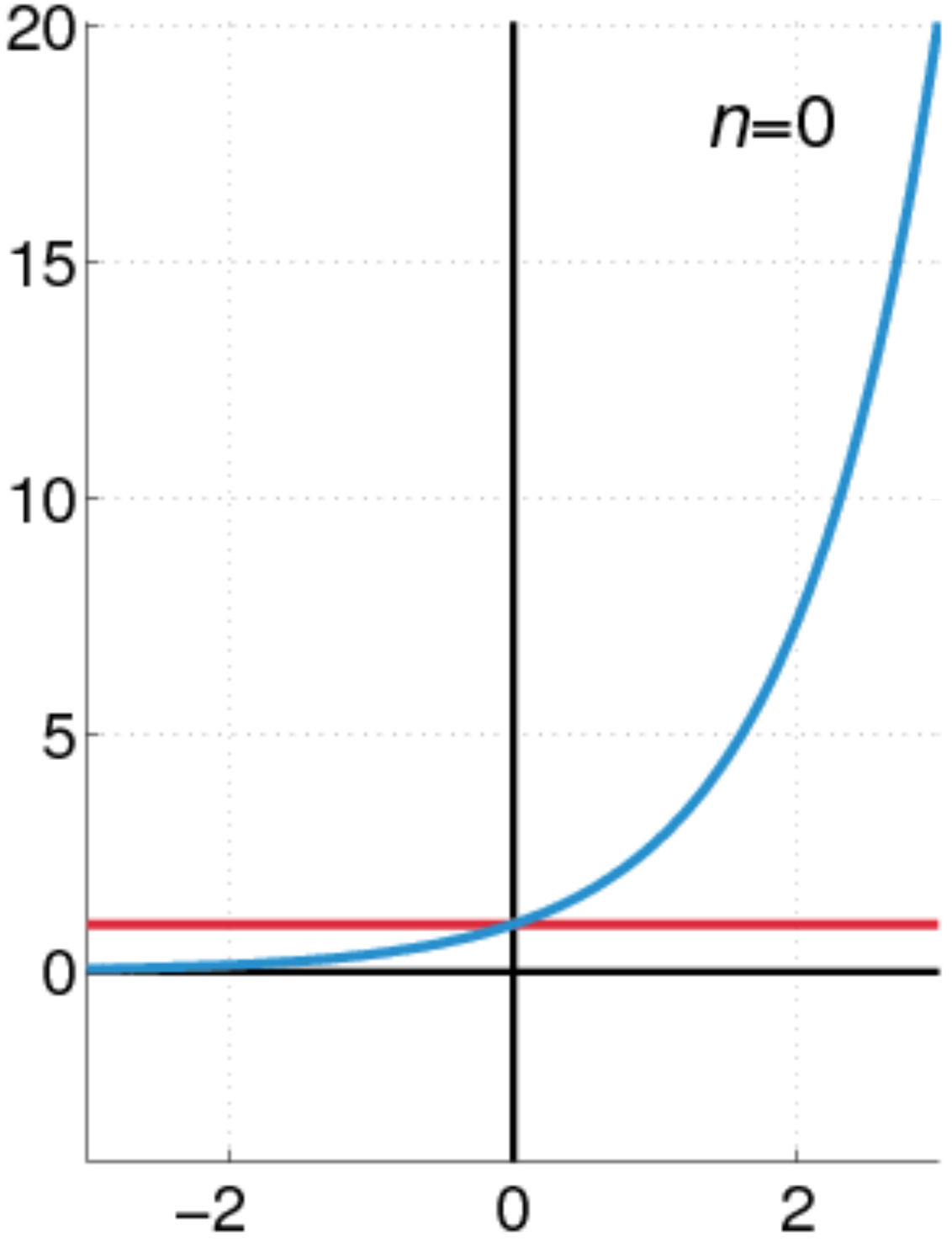


From wikipedia

Taylor series expansion: exponential function

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

Function: exp(x)
x0 = 0



From wikipedia

Whiteboard

- First-order and second-order gradient descent
- Big-O for these methods
- Understanding the Hessian and stepsize selection

Gradient descent

Algorithm 1: Batch Gradient Descent(Err, $\mathbf{X}, \mathbf{y}$)

1: // A non-optimized, basic implementation of batch gradient descent
2: $\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
3: err $\leftarrow \infty$
4: tolerance $\leftarrow 10e^{-4}$
5: $\alpha \leftarrow 0.1$
6: **while** $|\text{Err}(\mathbf{w}) - \text{err}| > \text{tolerance}$ **do**
7: err $\leftarrow \text{Err}(\mathbf{w})$
8: // The step-size α should be chosen by line-search
9: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \text{Err}(\mathbf{w}) = \mathbf{w} - \alpha \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$
10: **return** $\mathbf{w}$

Recall: for error function $E(\mathbf{w})$ goal is to solve $\nabla E(\mathbf{w}) = \mathbf{0}$

Convergence rates

- f(w_t) - f(w*) = O(1/t) (i.e., after t iterations) called **sublinear** convergence rate (and is not particularly good)
  - "The longer you run the algorithm, the less progress it makes"
  - To get within O(1/epsilon), need t = 1/epsilon iterations
- Linear convergence much better: t = 1/ln(epsilon)
  - Consistently makes progress
- Superlinear convergence awesome: t = 1/ln ln (epsilon)
  - Essentially a constant number of iterations

Convergence rates

- First order is sublinear, for more general f: f(w_t) - f(w*) < -C/ t
  - for some constant C that depends on the stepsize and initial error
  - Example: to get within epsilon accuracy, need t = C/epsilon steps

- First order is linear for “nice” f (Lipschitz and strongly convex)
  - Lipschitz means function does not change too quickly, locally
  - Strongly convex means function is not too flat

- Second order is superlinear! (for “nice” f)
  - to get within epsilon accuracy, need t = log(log(1/epsilon)) steps

Second-order

$\min f(x) := x^2 + e^x$

$$x^{k+1} = x^k - \frac{f'(x^k)}{f''(x^k)}$$

x	$f'(x)$
1	4.7182818
0	1
$-1/3$	.0498646
$-.3516893$	.00012
$-.3517337$	.00000000064

In addition, one more iteration gives $|f'(x^5)| \leq 10^{-20}$.

First-order

Many more iterations

k	x^k	$f(x^k)$	$f'(x^k)$
0	1	.37182818	4.7182818
1	0	1	1
2	$-.5$	.8565307	-0.3934693
3	$-.25$	.8413008	0.2788008
4	$-.375$	.8279143	$-.0627107$
5	$-.34075$	.8273473	.0297367
6	$-.356375$	.8272131	$-.01254$
7	$-.3485625$	.8271976	.0085768
8	$-.3524688$	.8271848	$-.001987$
9	$-.3514922$	.8271841	.0006528
10	$-.3517364$	.827184	$-.0000072$

Compared to

x	$f'(x)$
1	4.7182818
0	1
$-1/3$	.0498646
$-.3516893$	.00012
$-.3517337$	.00000000064

Gradient descent

Algorithm 1: Batch Gradient Descent(Err, $\mathbf{X}, \mathbf{y}$)

1: // A non-optimized, basic implementation of batch gradient descent
2: $\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
3: err $\leftarrow \infty$
4: tolerance $\leftarrow 10e^{-4}$
5: $\alpha \leftarrow 0.1$
6: **while** $|\text{Err}(\mathbf{w}) - \text{err}| > \text{tolerance}$ **do**
7: err $\leftarrow \text{Err}(\mathbf{w})$
8: // The step-size α should be chosen by line-search
9: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \text{Err}(\mathbf{w}) = \mathbf{w} - \alpha \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$
10: **return** $\mathbf{w}$

Recall: for error function $E(\mathbf{w})$ goal is to solve $\nabla E(\mathbf{w}) = \mathbf{0}$

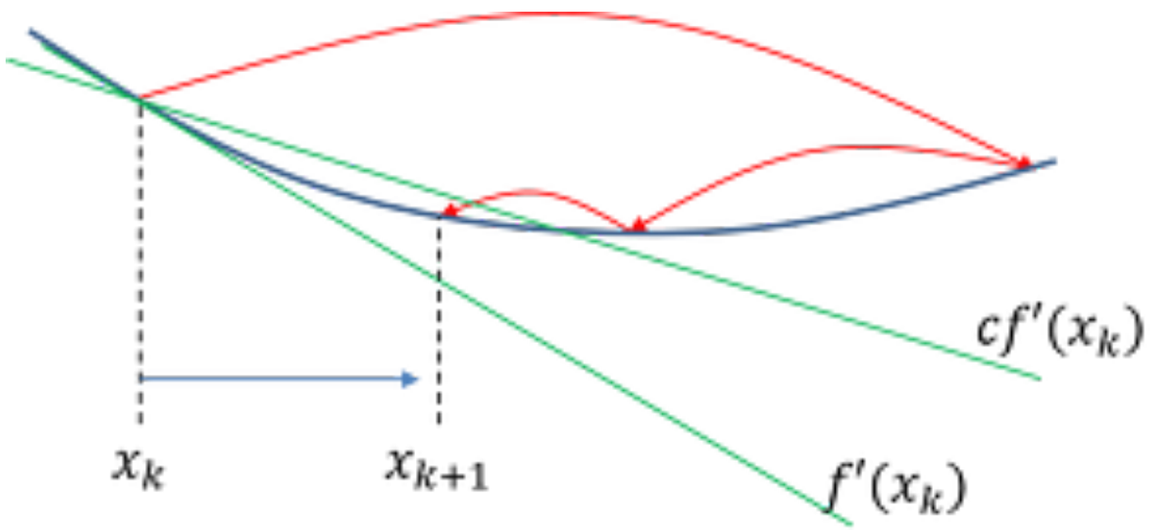
Line search

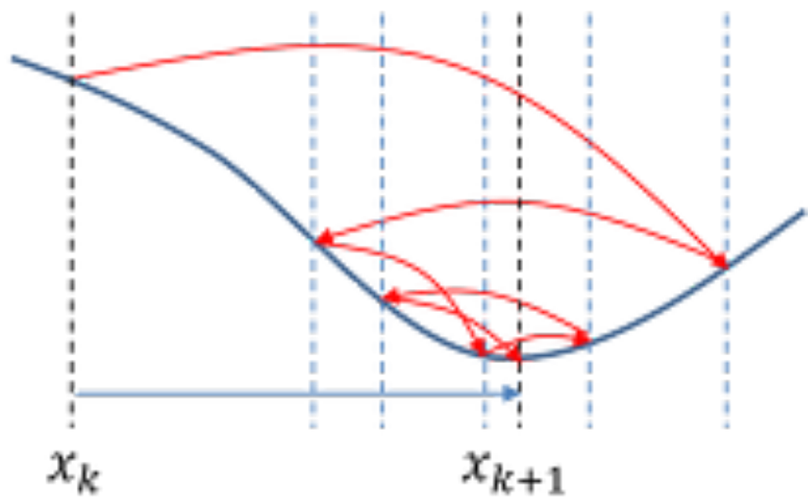
Want step-size such that

$$\alpha = \arg\min_{\alpha} E(\mathbf{w} - \alpha \nabla E(\mathbf{w}))$$

Backtracking line search:

1. Start with relatively large α (say $\alpha = 1$)
2. Check if $E(\mathbf{w} - \alpha \nabla E(\mathbf{w})) < E(\mathbf{w})$
3. If yes, use that α
4. Otherwise, decrease α (e.g., $\alpha = \alpha/2$), and check again





Intuition for first and second order

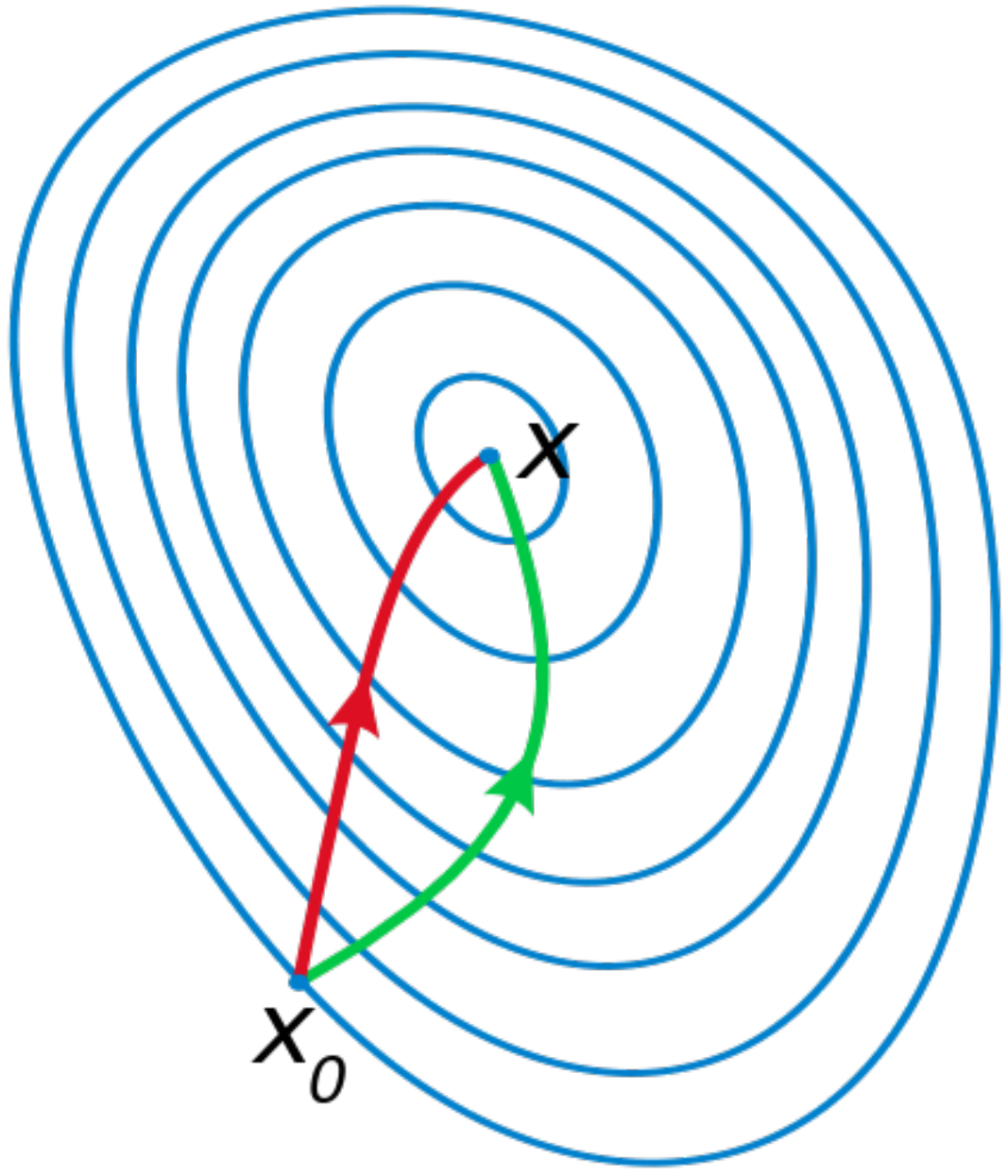
- Locally approximate function at current point

- First order: locally approximate with rough estimate of quadratic and step in the direction of the minimum of that approximation

- For second order, locally approximate as quadratic and step in the direction of the minimum of that quadratic function

- What happens if the true function is quadratic?

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \left(H_{f(\mathbf{x}^{(i)})}\right)^{-1} \cdot \nabla f(\mathbf{x}^{(i)}),$$



Newton in red

What is the second-order linear regression update?

Algorithm 1: Batch Gradient Descent$(\text{Err}, \mathbf{X}, \mathbf{y})$

// A non-optimized, basic implementation of batch gradient descent
$\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
$\text{err} \leftarrow \infty$
$\text{tolerance} \leftarrow 10e^{-4}$
$\alpha \leftarrow 0.1$
while $|\text{Err}(\mathbf{w}) - \text{err}| > \text{tolerance}$ **do**
    $\text{err} \leftarrow \text{Err}(\mathbf{w})$
    // The step-size α should be chosen by line-search
    $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \text{Err}(\mathbf{w}) = \mathbf{w} - \alpha \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$
return $\mathbf{w}$

$$\mathbf{g} = \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$$
$$\mathbf{H} = \mathbf{X}^\top \mathbf{X}$$
$$\mathbf{H}^{-1}\mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$$
$$= \mathbf{H}^{-1}\mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{H}^{-1}\mathbf{X}^\top \mathbf{y}$$
$$= \mathbf{w} - (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$$

First step: $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1}\mathbf{g}$

$$\implies \mathbf{w} \leftarrow \mathbf{w} - \left(\mathbf{w} - (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}\right)$$
$$\implies \mathbf{w} \leftarrow (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$$

Quasi-second order methods

- Approximate inverse Hessian, can be much more efficient
  - Imagine if you only kept the diagonal of the inverse Hessian
  - How expensive would this be?
- Examples: LBFGS, low-rank approximations, Adagrad, Adadelta, Adam

Batch optimization

- What are some issues with batch gradient descent?
- When might it be slow?
- Recall: O(d n) per step

Stochastic gradient descent

Algorithm 2: Stochastic Gradient Descent$(E, \mathbf{X}, \mathbf{y})$

1: $\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
2: **for** $t = 1, \ldots, n$ **do**
3: // For some settings, we need the step-size α_t to decrease with time
4: $\mathbf{w} \leftarrow \mathbf{w} - \alpha_t \nabla E_t(\mathbf{w}) = \mathbf{w} - \alpha_t(\mathbf{x}_t^\top \mathbf{w} - y_t)\mathbf{x}_t$
5: **end for**
6: **return** $\mathbf{w}$

For batch error: $\hat{E}(\mathbf{w}) = \sum_{t=1}^{n} E_t(\mathbf{w})$
e.g., $E_t(\mathbf{w}) = (\mathbf{x}_t^\top \mathbf{w} - y_t)^2$
$\hat{E}(\mathbf{w}) = \sum_{t=1}^{n} E_t(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$
$\nabla \hat{E}(\mathbf{w}) = \sum_{t=1}^{n} \nabla E_t(\mathbf{w})$
$E(\mathbf{w}) = \int_{\mathcal{X}} \int_{\mathcal{Y}} f(\mathbf{x}, y)(\mathbf{x}^\top \mathbf{w} - y)^2 dy d\mathbf{x}$

- Stochastic gradient descent (stochastic approximation) minimizes with an unbiased sample of the gradient $\mathbb{E}[\nabla E_t(\mathbf{w})] = \nabla E(\mathbf{w})$

Batch gradient unbiased sample of true gradient

$$\mathbb{E}\left[\frac{1}{n}\nabla\hat{E}(\mathbf{w})\right] = \frac{1}{n}\mathbb{E}\left[\sum_{i=1}^{n}\nabla E_i(\mathbf{w})\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[\nabla E_i(\mathbf{w})]$$

$$\text{e.g., } \mathbb{E}[(\boldsymbol{X}_i^\top\mathbf{w} - Y_i)\boldsymbol{X}_i]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\nabla E(\mathbf{w})$$

$$= \nabla E(\mathbf{w})$$

Stochastic gradient descent

- Can also approximate gradient with more than one sample (e.g., mini-batch), as long as $\mathbb{E}[\nabla E_t(\mathbf{w})] = \nabla E(\mathbf{w})$

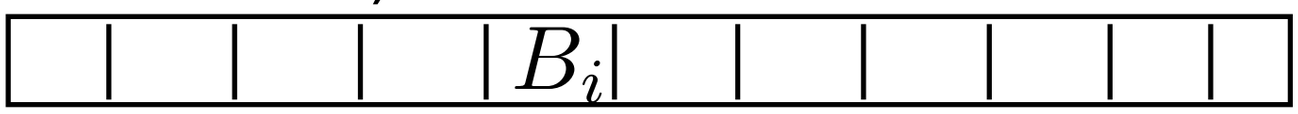
- Proof of convergence and conditions on step-size: Robbins-Monro ("A Stochastic Approximation Method", Robbins and Monro, 1951)

- A big focus in recent years in the machine learning community; many new approaches for improving convergence rate, reducing variance, etc.

Gradient descent approaches

- Commonly use stochastic gradient descent (SGD) or mini-batch SGD, for a relatively small mini-batch size of say 32

- Mini-batch: update weights using an averaged gradient over a subset of 32 samples (a mini batch B)



- Approach: for one epoch (iterating over the dataset once)
  - SGD with one sample: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla c_i(\mathbf{w}_t)$
  - SGD with mini-batch: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \sum_{j \in B_i} \nabla c_j(\mathbf{w}_t)$
- If n = 1000, mini-batch b = 10, how many iterations for SGD and mini-batch SGD within one epoch?

How do we pick the stepsize?

- Less clear than for batch gradient descent

- Basic algorithm, the step sizes must decrease with time, but be non-negligible in magnitude (e.g., 1/t)

$$\sum_{i=1}^{\infty} \alpha_t^2 < \infty$$

$$\sum_{i=1}^{\infty} \alpha_t = \infty$$

- Recent further insights into improving selection of stepsizes, and reducing variance (e.g., SAGA, SVG)

- Note: look up stochastic approximation as alternative name

What are the benefits of SGD?

- For batch gradient descent: to get w such that f(w) - f(w*) < epsilon, need O(ln(1/epsilon)) iterations
  - with conditions on f (strongly convex, gradient Lipschitz continuous)
  - 1 iteration of GD for linear regression: $\mathbf{w} = \mathbf{w} - \alpha_t \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$
  - ln(1/0.001) approx= 7

$$= \mathbf{w} - \alpha_t \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i)\mathbf{x}_i$$

- For stochastic gradient descent: to get w such that f(w) - f(w*) < epsilon, need O(1/epsilon) iterations
  - with conditions on f_i (strongly convex, gradient Lipschitz continuous)
  - 1 iteration of SGD for linear regression: $\mathbf{w} = \mathbf{w} - \alpha_t (\mathbf{x}^\top \mathbf{w} - y_t)\mathbf{x}_t$
  - 1/0.001 = 1000

Whiteboard

- **Exercise**: derive an algorithm to compute the solution to l1-regularized linear regression (i.e., MAP estimation with a Gaussian likelihood p(y l x, w) and Laplace prior)
  - First write down the Laplacian
  - Then write down the MAP optimization
  - Then determine how to solve this optimization
- Next: Generalized linear models