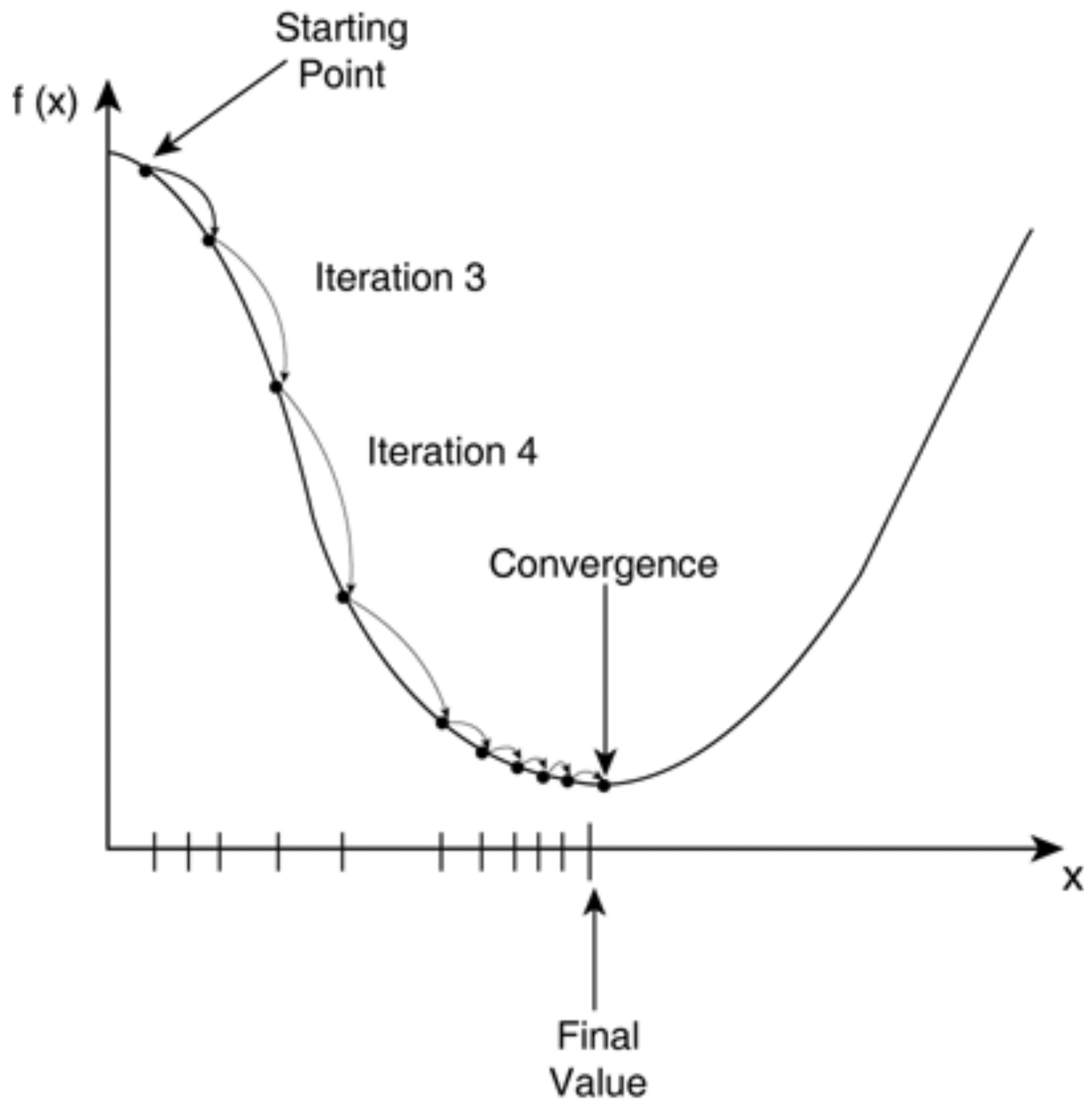
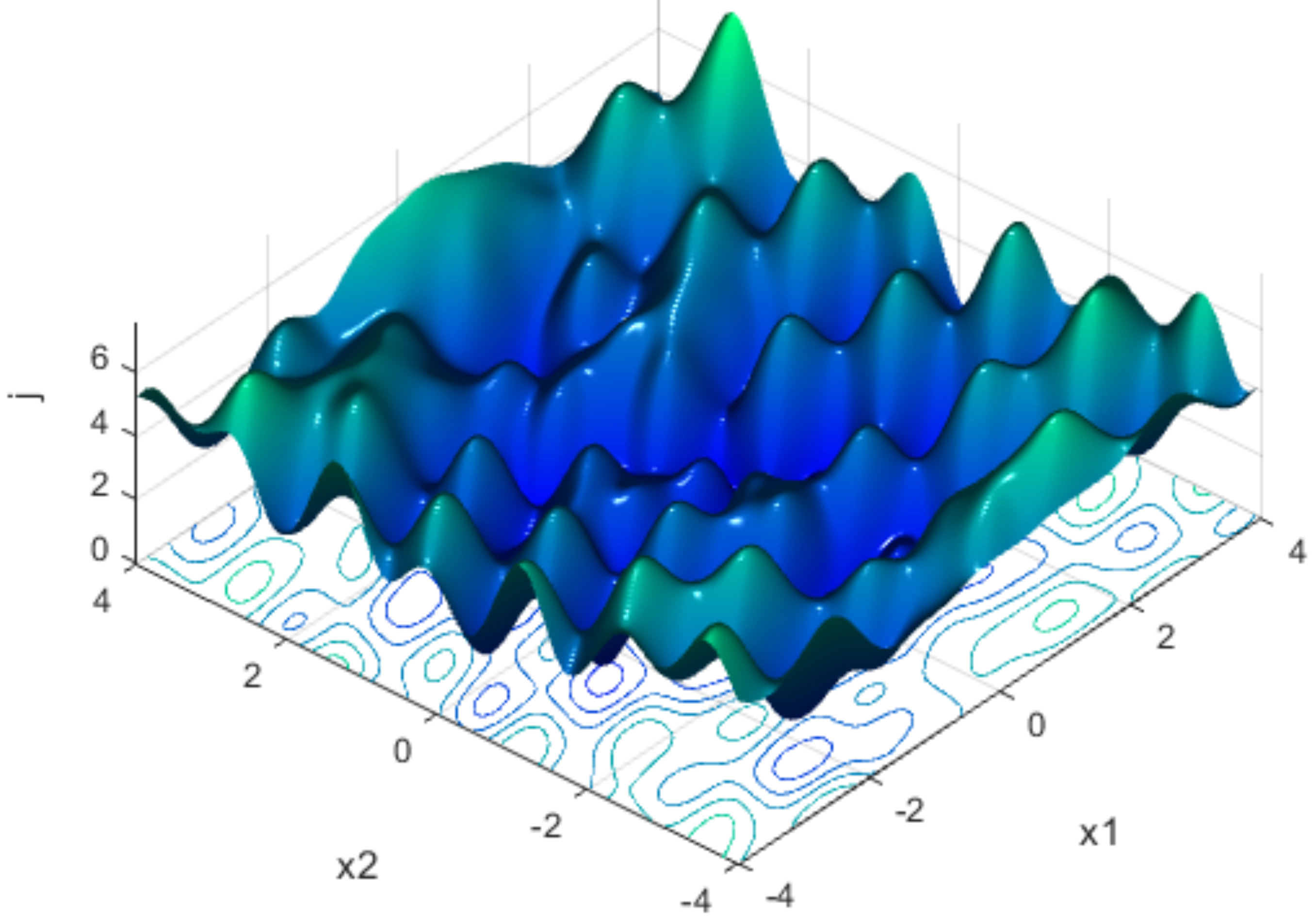
Optimization

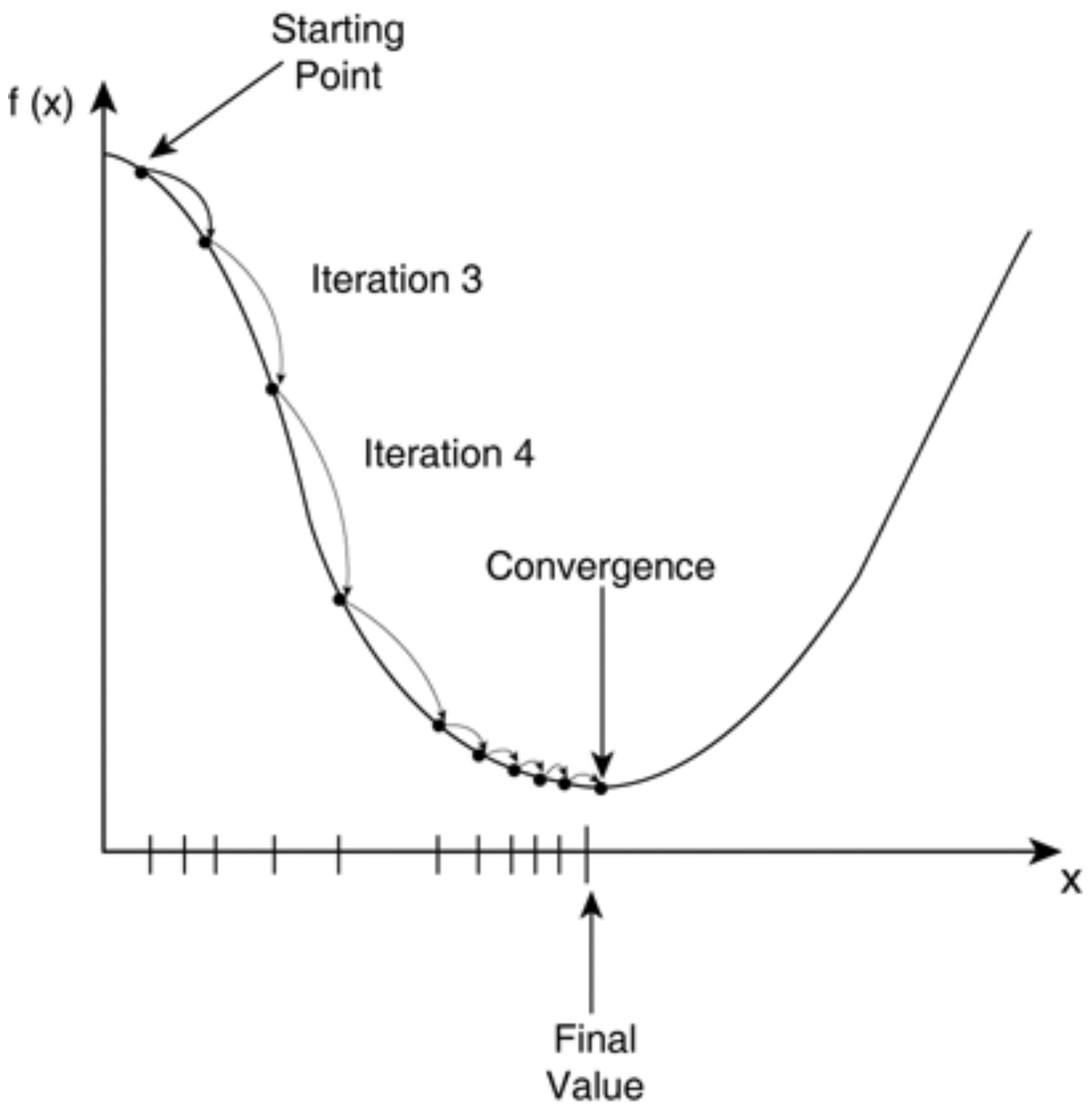




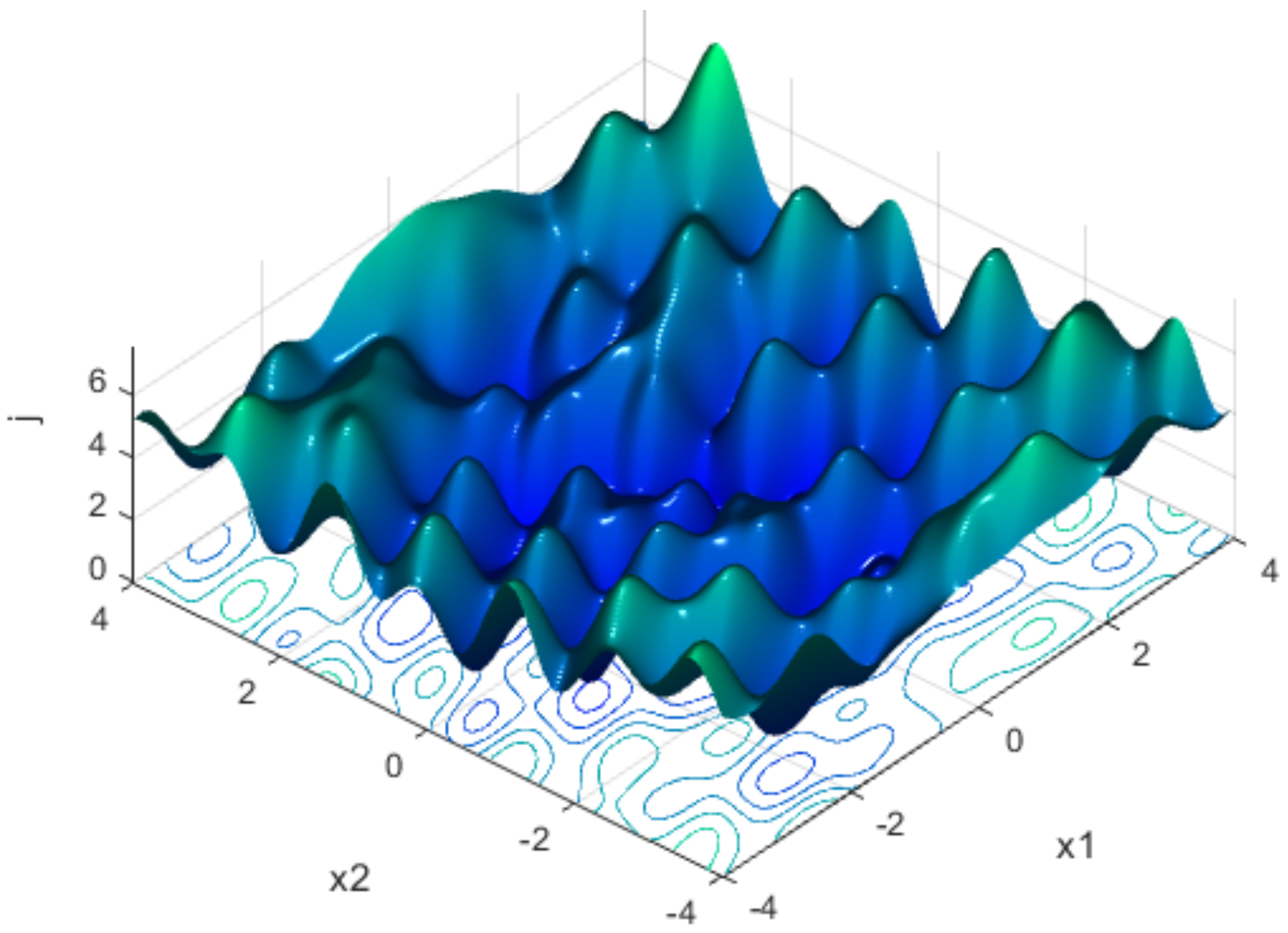
Why should you care about the solution strategies?

- 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!

Gradient descent intuition



Convex function



Non-convex function

Gradient descent

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

```
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 $|\mathrm{Err}(\mathbf{w}) - \mathrm{err}| >$ tolerance do
7:     err $\leftarrow \mathrm{Err}(\mathbf{w})$
8:     // The step-size $\alpha$ should be chosen by line-search
9:     $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \mathrm{Err}(\mathbf{w}) = \mathbf{w} - \alpha \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$
10: return $\mathbf{w}$
```

Comments (Oct 3, 2017)

- Assignment 1 marks released
- Thought questions 1 marks released
  - Really fun questions!
- Assignment 2 released
  - Heavier on programming (in python)
  - Provided python code framework
- Lab this week?

Mean and mode

- Mean is the average value, under p(x)
  - As mentioned, expected value seems like the wrong (since its the not the value you necessarily expect to see), but such is life
- Mode is the most likely value, according to p(x)

Purpose of the prior

- “For the MAP estimate, what is the purpose of the prior once we observe this data?
- Typical (invalid) answer: “the number of samples increase, the MAP estimate converges to the ML estimate”
  - Technically correct, but not answering the question

Python pitfalls

- Pass by assignment (or the myriad of other terms):
  - pass reference to object, and change inside function (e.g., list.append("new value"), or a[0] = 1), then changes outside function
  - if overwrite reference (e.g., a = np.zeros((1,1))), then 'a' now points to new object, and any changes to 'a' does not change the old object
  - Think of 'a' as a pointer
- Broadcasting versus dot-product multiplication
  - numpy.multiply and * do element-wise multiplication, and allow multiplication of variables that are not of the same dimension (which can sometimes make it hard to find bugs, since it "works")
  - numpy.dot does matrix multiplication
- You do not have to use the provided code, but should help

Python code

- Main script_regression.py runs algorithms on a dataset
    - loads data
    - splits data
    - gather errors from running algorithms
- regressionalgorithms.py contains algorithm code
    - parent class Regressor
    - all regression algorithms are child classes
- utils.py contains some useful additional functions
    - But most of the functions in utils.py will not be useful yet

Where does gradient descent come from?

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

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?

Existence

- Does a solution for MLE always exist?
- Is it unique? Is it identifiable?
  - A collection of models are identifiable if a parameter uniquely determine a model,
  - i.e., if model-one = model-two, then parameter-one = parameter-two
  - e.g., if Gaussian(mu1, sigma) = Gaussian (mu2, sigma), then we know mu1 = mu2

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
- How do we pick a good starting point for gradient descent?

Taylor series expansion

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

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

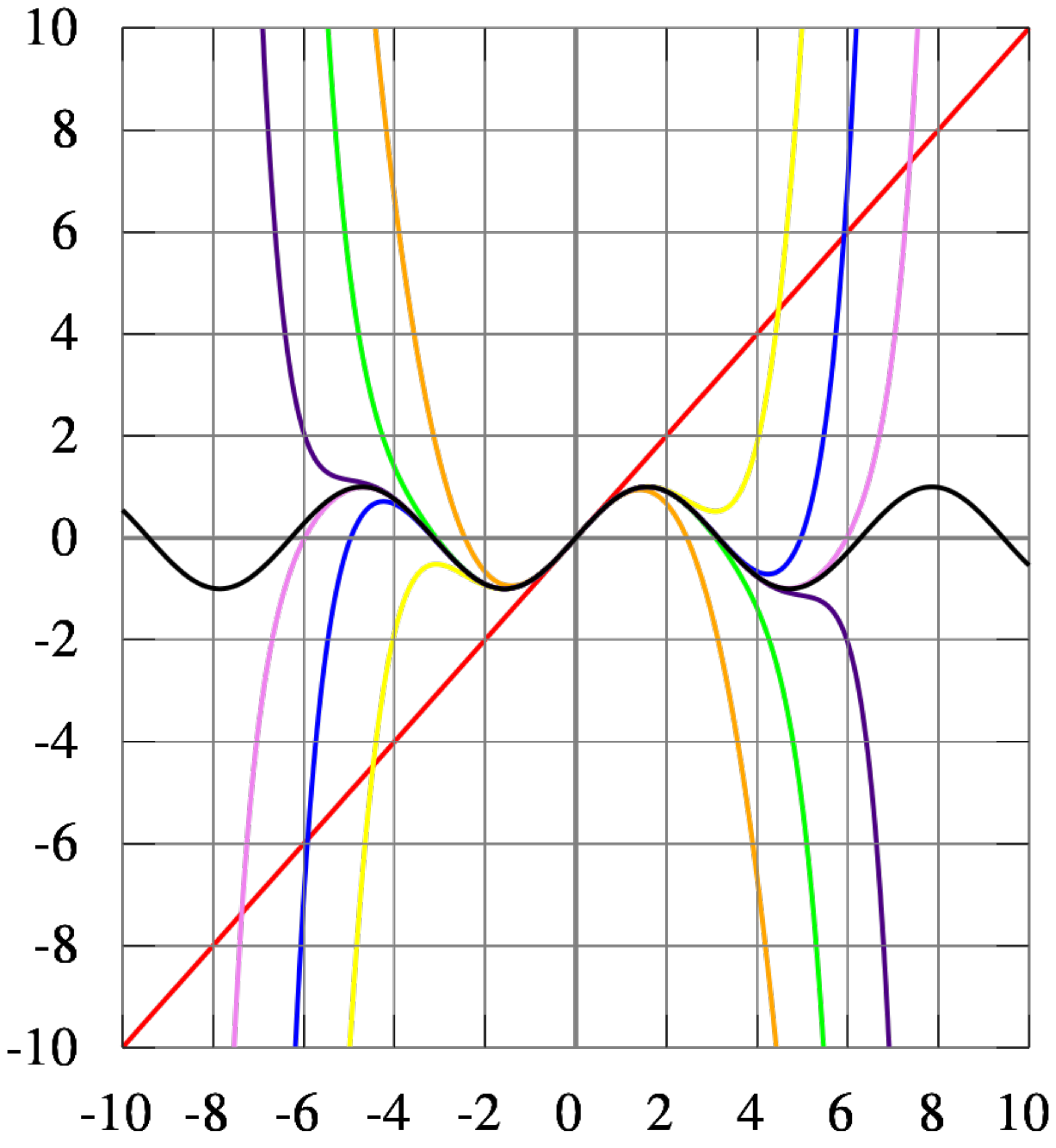
where $f^{(n)}(x_0)$ is the n-th derivative of function $f(x)$ evaluated at point x_0.

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

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

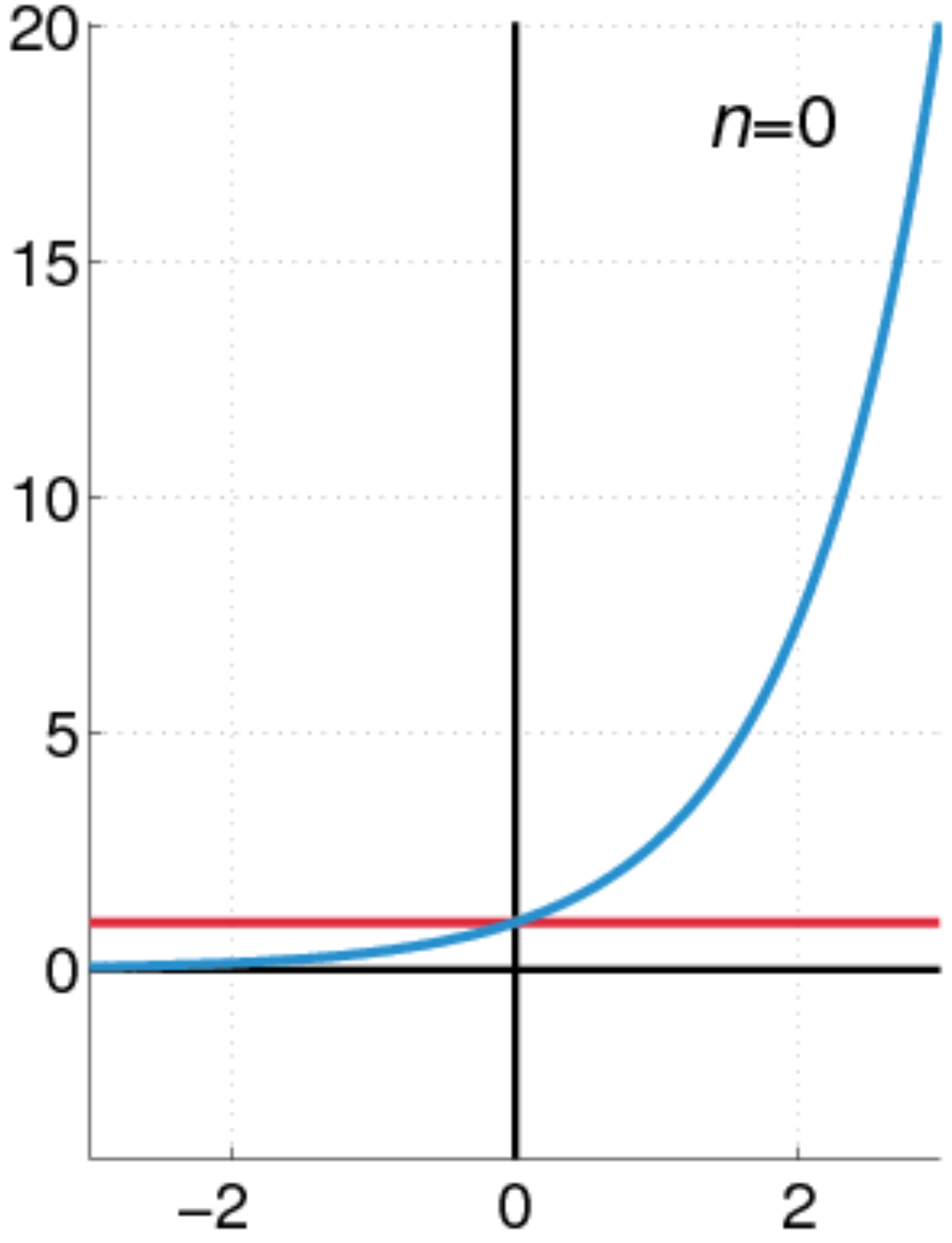


degree **1**, **3**, **5**, **7**, **9**, **11** and **13**.

From wikipedia

Taylor series expansion

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



From wikipedia

Multivariate Taylor series

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \cdot (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \cdot H_{f(\mathbf{x}_0)} \cdot (\mathbf{x} - \mathbf{x}_0),$$

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_k} \right)$$

$$H_{f(\mathbf{x})} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_k} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & & \\ \vdots & & \ddots & \\ \frac{\partial^2 f}{\partial x_k \partial x_1} & & & \frac{\partial^2 f}{\partial x_k^2} \end{bmatrix}$$

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$(\mathrm{Err}, \mathbf{X}, \mathbf{y})$

```
 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 $|\mathrm{Err}(\mathbf{w}) - \mathrm{err}| >$ tolerance do
 7:     err $\leftarrow \mathrm{Err}(\mathbf{w})$
 8:     // The step-size $\alpha$ should be chosen by line-search
 9:     $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \mathrm{Err}(\mathbf{w}) = \mathbf{w} - \alpha \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$
10: return w
```

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

Comments (Oct. 5)

- Mini-project: first step is to find a dataset

Thought question

- “Does maximum likelihood always given the underlying distribution, in the limit?”

- Yes, if the distribution class we chose includes the true (underlying) distribution

Thought question

- “In page 41, it states that in the presence of a large dataset, the two approaches converge to the same model so long as "the prior does not have zero probability on f". If both models converge to the same value given increasing amounts of data, does it not become useful to understand how quickly either method approaches the true value as data increases. Since any realistic problem will have limiations on the amount of data given, how would an application working with limited data that may be slow or incovenient to collect choose which method of estimation, MAP or ML, converges to the true model more quickly?”

- **Consistency** is asymptotic, **convergence rate** is for finite sample analysis (e.g., error decreases as O(1/n))

- Would you expect MAP or ML to have a better rate?

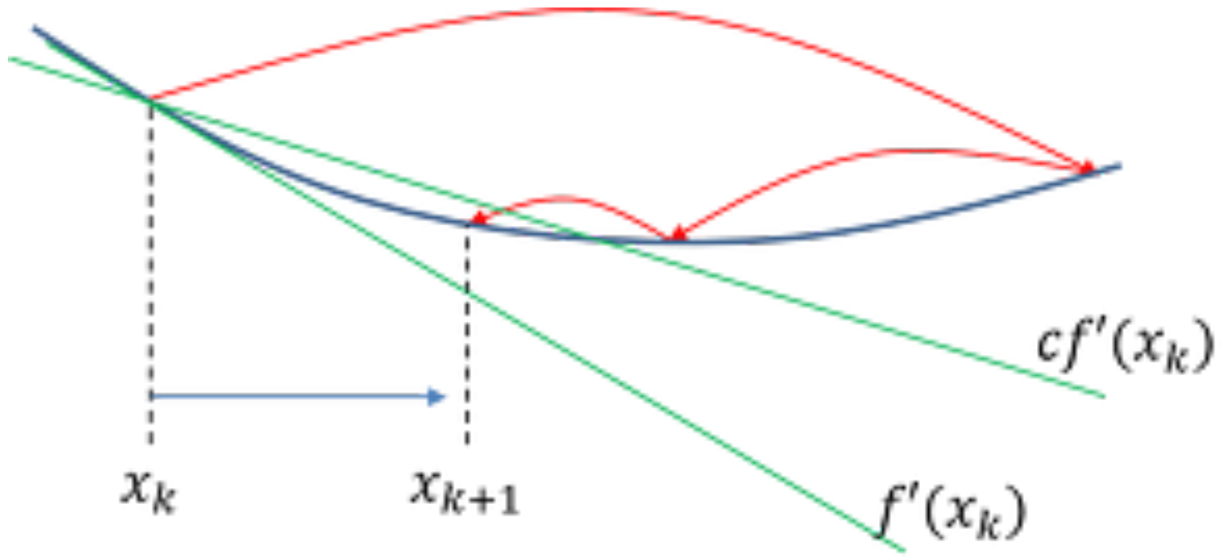
Line search

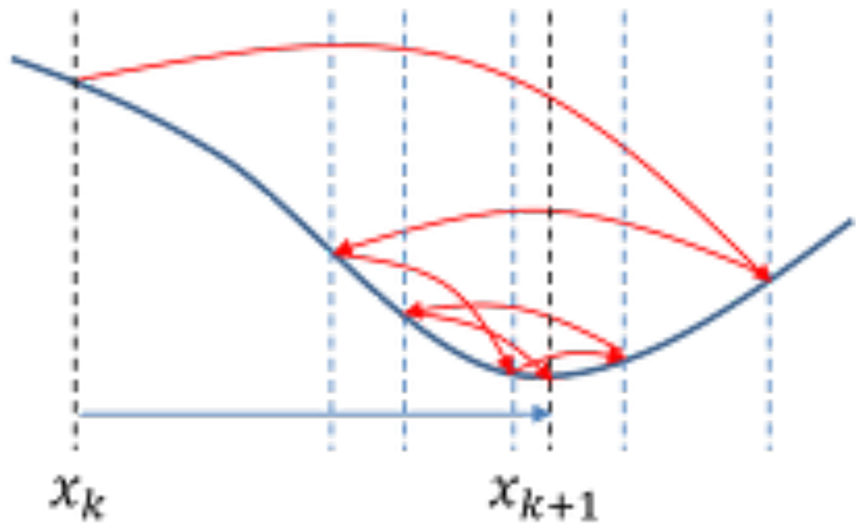
Want step-size such that

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

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





What is the second-order gradient descent update?

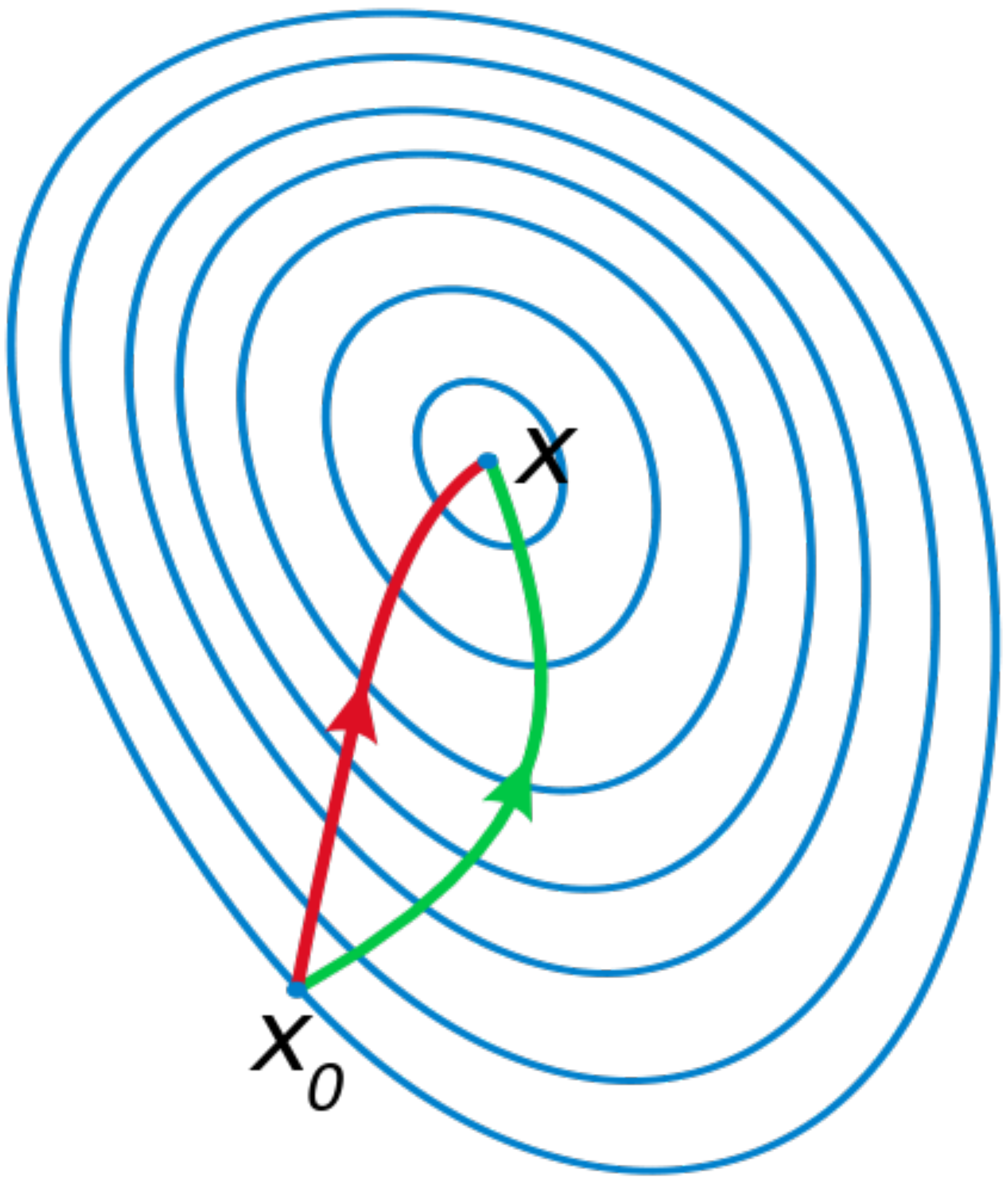
Algorithm 1: Batch Gradient Descent(Err, $\mathbf{X}, \mathbf{y}$)

```
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 |Err($\mathbf{w}$) − err| > tolerance do
7:     err $\leftarrow$ Err($\mathbf{w}$)
8:     // The step-size $\alpha$ should be chosen by line-search
9:     $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \mathrm{Err}(\mathbf{w}) = \boxed{\mathbf{w} - \alpha \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y})}$
10: return $\mathbf{w}$
```

Intuition for first and second order

- Locally approximate function at current point
- For first order, locally approximate as linear and step in the direction of the minimum of that linear function
- For second order, locally approximate as quadratic and step in the direction of the minimum of that quadratic function
  - a quadratic approximation is more accurate
- 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

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 $\alpha_t$ 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.

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

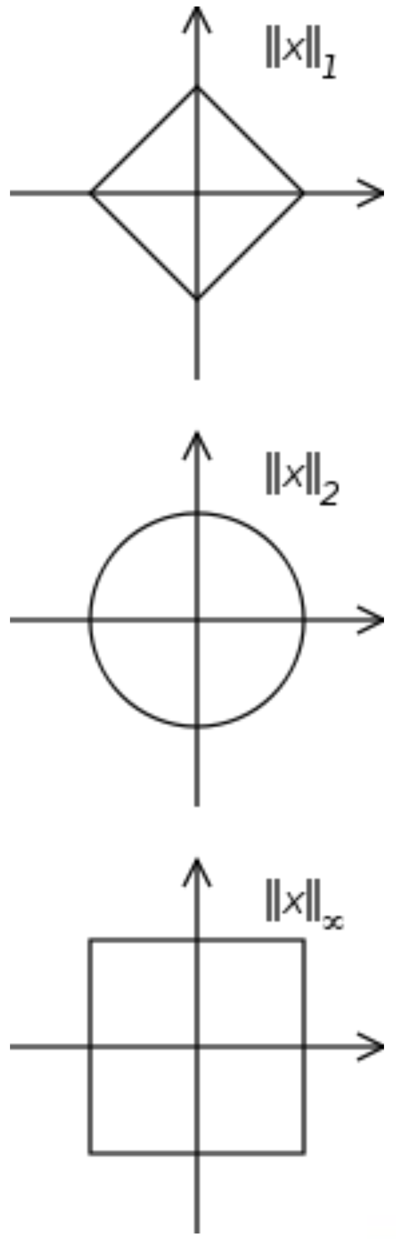
What are the benefits of SGD?

- For batch gradient descent: to get w such that f(w) - f(w*) < epsilon, need O(ln(1/epsilon)) iterations
  - with conditions on f (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

Regularization intuition



$$\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1$$

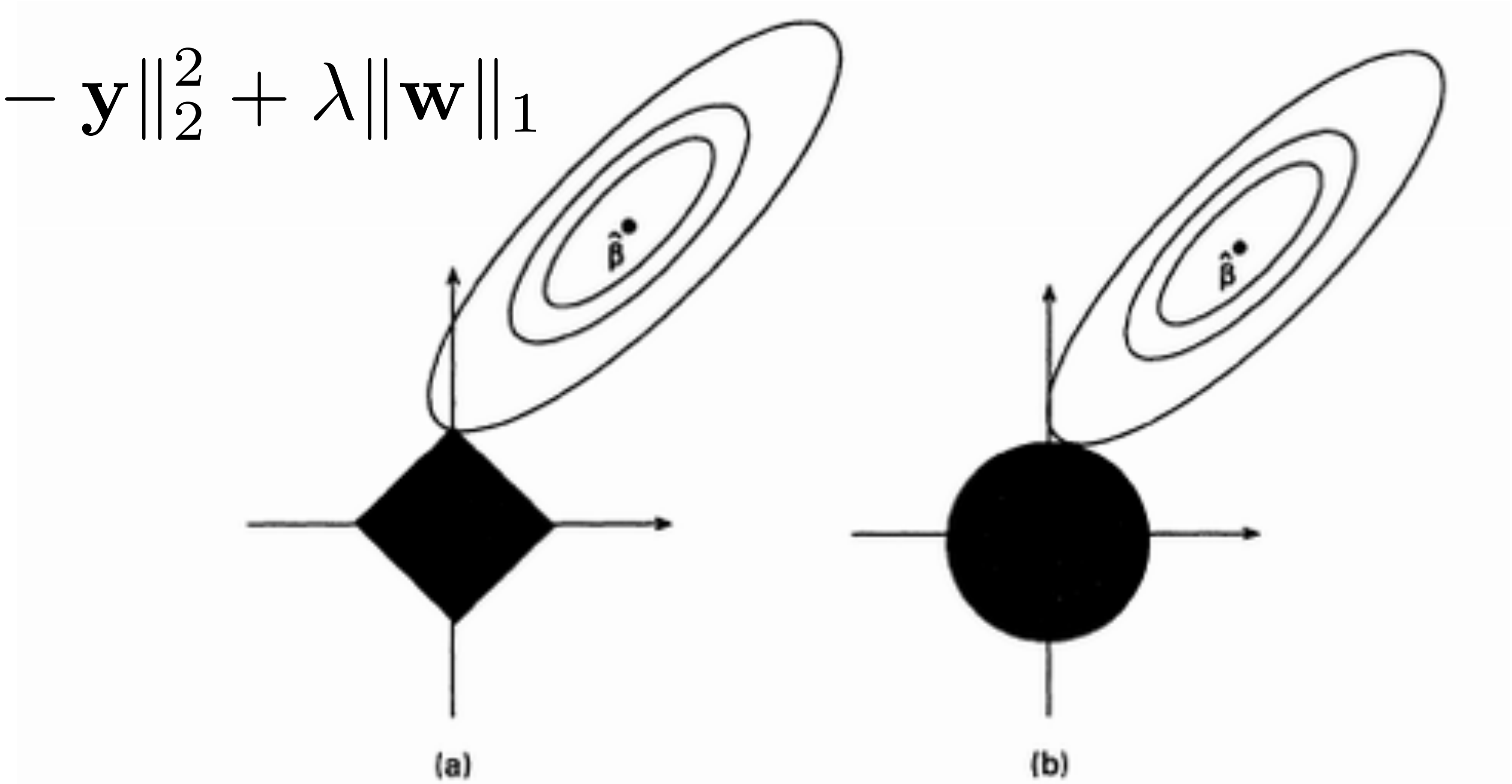
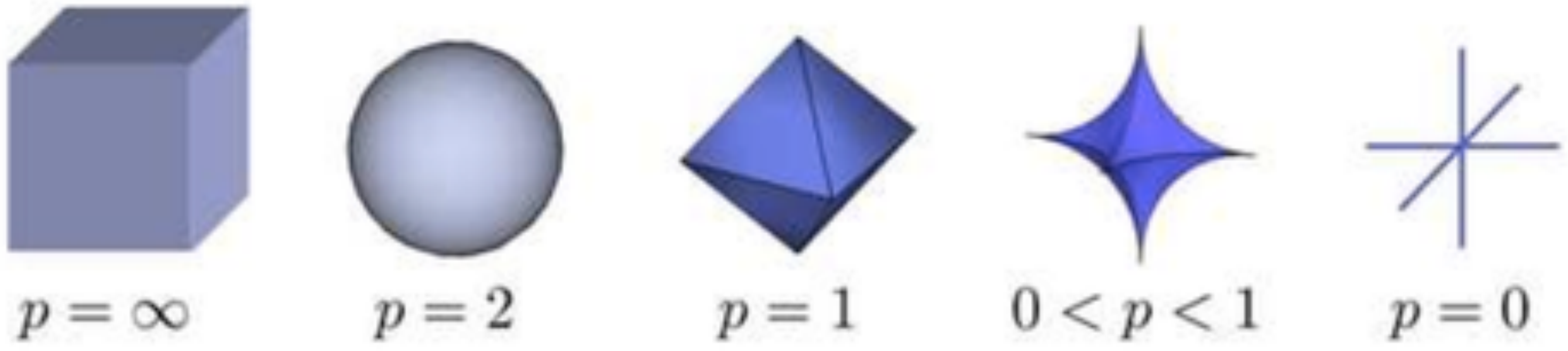


Fig. 2. Estimation picture for (a) the lasso and (b) ridge regression



Why regularize?

$$\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2$$

- Why would we a priori believe our weights should be close to zero? What if one of our coefficients needs to be big?
- What happens if one magnitude of the features is really big and another is small?
  - e.g., x1 = house price (100000), x2 = number of rooms (3)
- What is the disadvantage to regularizing? What does it do to the weights?

Why would we do feature selection?

- Why not use all the features? It is more information?
- What settings might you care to do feature selection?
- In such setting, do you see any issues with using Lasso?
  - Where Lasso is proximal gradient descent

Whiteboard

- **Exercise**: derive an algorithm to compute the solution to l1-regularized linear regression (i.e., MAP estimation with a Gaussian likelihood p(y | 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