COMP0130: Robotic Vision and Navigation

Lecture 06D: Inference and Optimization

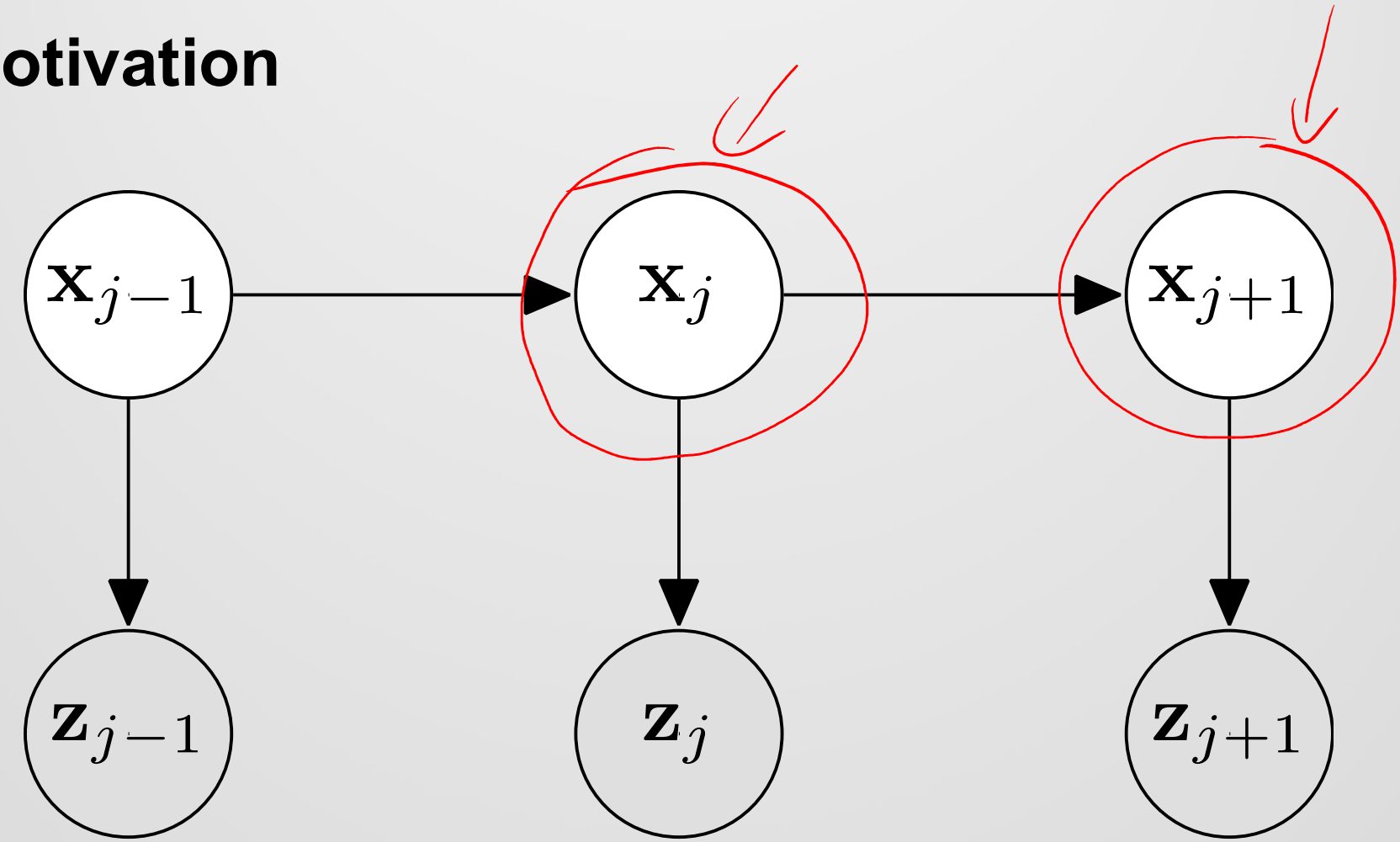
Structure

- Motivation
- The Problem with Means
- Maximum Aposteriori Estimation
- Optimization Algorithms
- Approximate Covariance Calculations

Motivation

- The factor graph stores the entire probability density in a single equation
- Theoretically, we can ask any question we like of it and compute the answer
- In robotics, we are often interested in computing a point estimate which we can use in control

Motivation


$\mathbf{x}_{j-1}$
$\mathbf{x}_j$
$\mathbf{x}_{j+1}$
$\mathbf{z}_{j-1}$
$\mathbf{z}_j$
$\mathbf{z}_{j+1}$

Expected Values

- *Motivation*
- Expected Values
- *Maximum Aposteriori Estimation*
- *Optimization Algorithms*
- *Approximate Covariance Calculations*

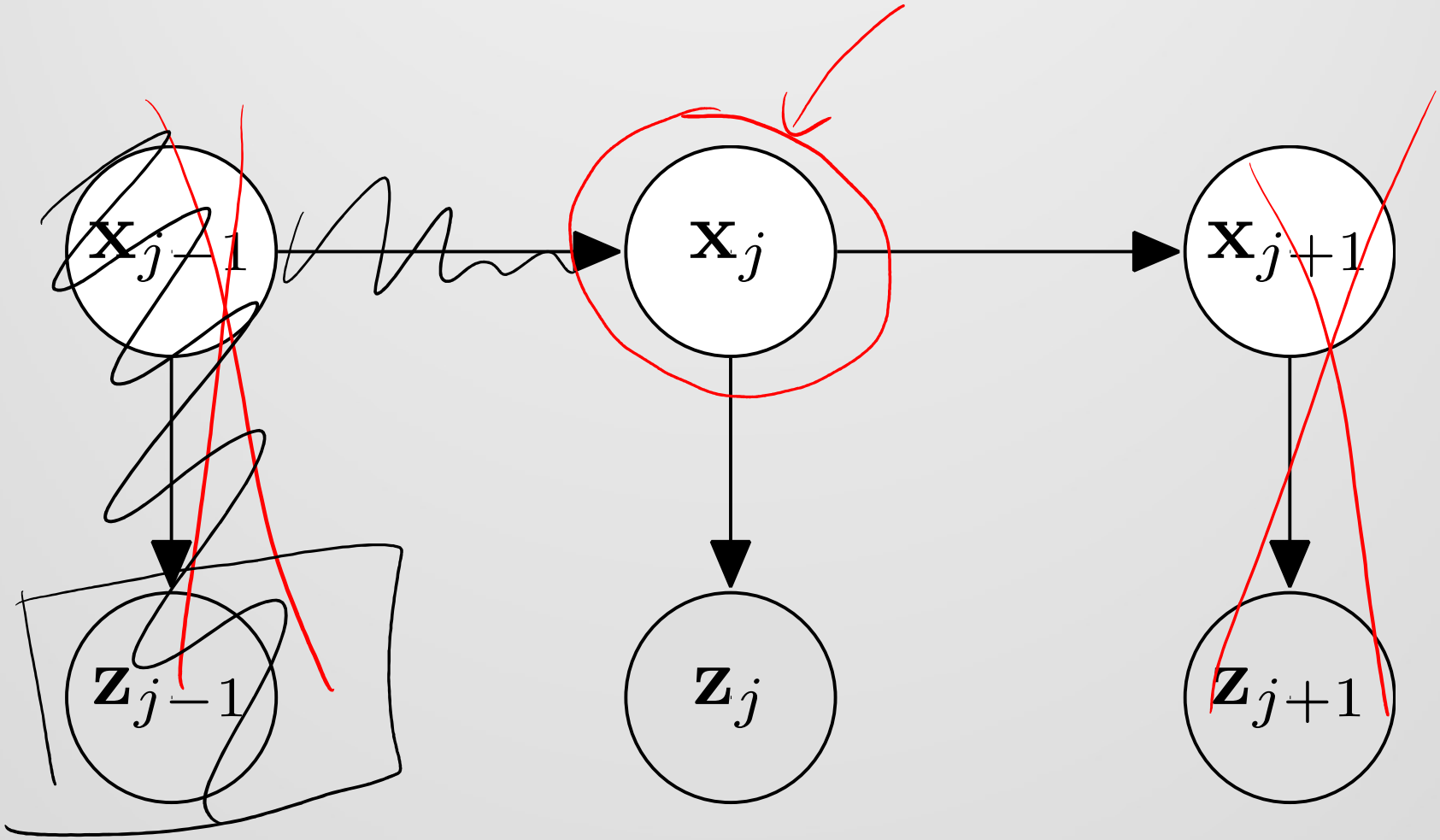
Computing the Mean

- Suppose we want to compute the mean of the jth state
- The mean is given by

$$\mathbb{E}\left[\mathbf{x}_j\right] = \int \mathbf{x}_j f\left(\mathbf{x}_j | \mathbb{I}_k\right) \mathrm{d}\mathbf{x}_j$$

- To do this, we have to marginalize

Computing the Mean



Marginalization

- Marginalization is the “proper” way to remove variables from a graph
- We do this by integrating all the other variables out

$$f\left(\mathbf{x}_j|\mathbb{I}_k\right) = \int f\left(\mathbf{x}_{1:k}|\mathbb{I}_k\right)\mathrm{d}\mathbf{x}_{1:k\backslash j}$$

$$\overbrace{1:k}\backslash j = [1, 2, \cdots, j-1, j+1, \cdots, k]$$

Computed Expected Values

- More generally, we can specify any kind of function,

$$\mathbb{E}\left[\mathbf{b}\left[\mathbf{x}_{1:k}\right]\right] = \int \mathbf{b}\left[\mathbf{x}_{1:k}\right] f\left(\mathbf{x}_{1:k} | \mathbb{I}_k\right) \mathrm{d}\mathbf{x}_{1:k}$$

- This can be used to compute things like covariances and cross correlations as well

Computing the Mean

- However, performing the marginalization is awful
- We have to do all the integration work we were desperately trying to avoid
- Therefore, we need to look at other approaches

Structure

- *Motivation*
- *The Problem with Means*
- Maximum Aposteriori Estimation MAP
- *Optimization Algorithms*
- *Approximate Covariance Calculations*

Maximum Aposteriori Estimation

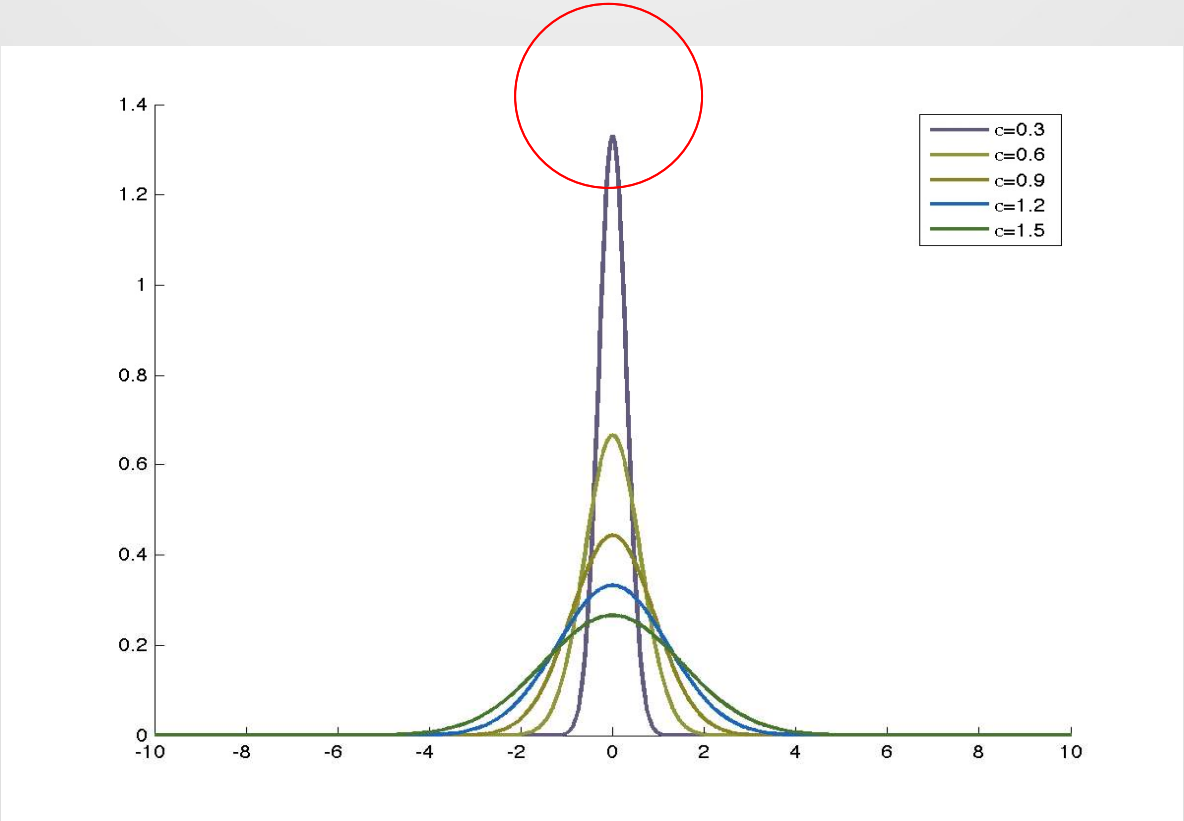
- The problem with estimating the mean is that we have to integrate over the entire distribution, which we can't do in practice
- An alternative idea is to just extract information directly from the distribution without integration
- One particularly easy approach is to do peak finding

Maximum Aposteriori Estimation

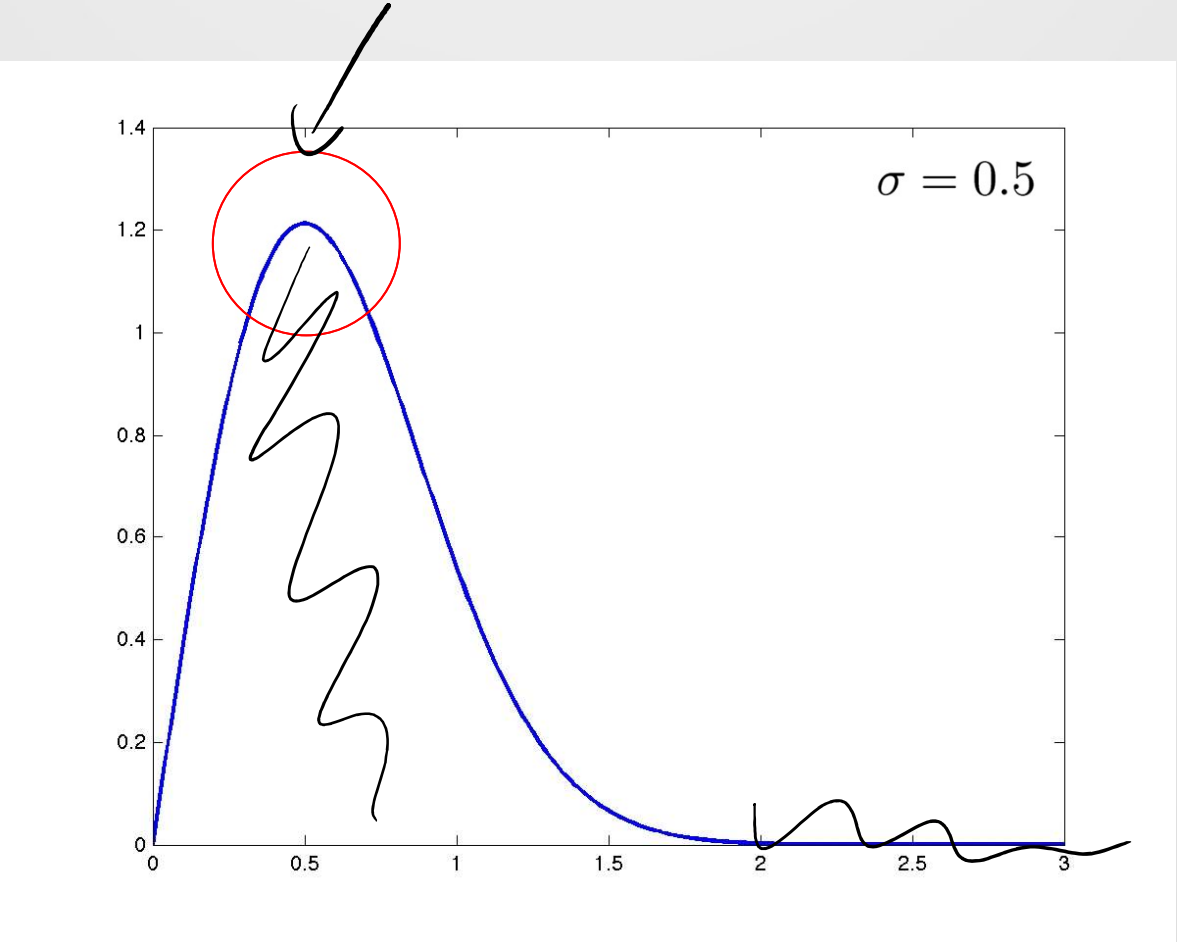
- Define the output estimate to be

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} f(\mathbf{x})$$

MAP of a Gaussian PDF



MAP of a Rayleigh PDF



Maximum Aposteriori Estimation

- Define the output estimate to be

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} f(\mathbf{x})$$

- Advantages:
  - We do not need to know the normalization constant (no integration!)
  - The estimate itself is often pretty sensible

Normalization Constant not Required

- Suppose we have the solution

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} f(\mathbf{x})$$

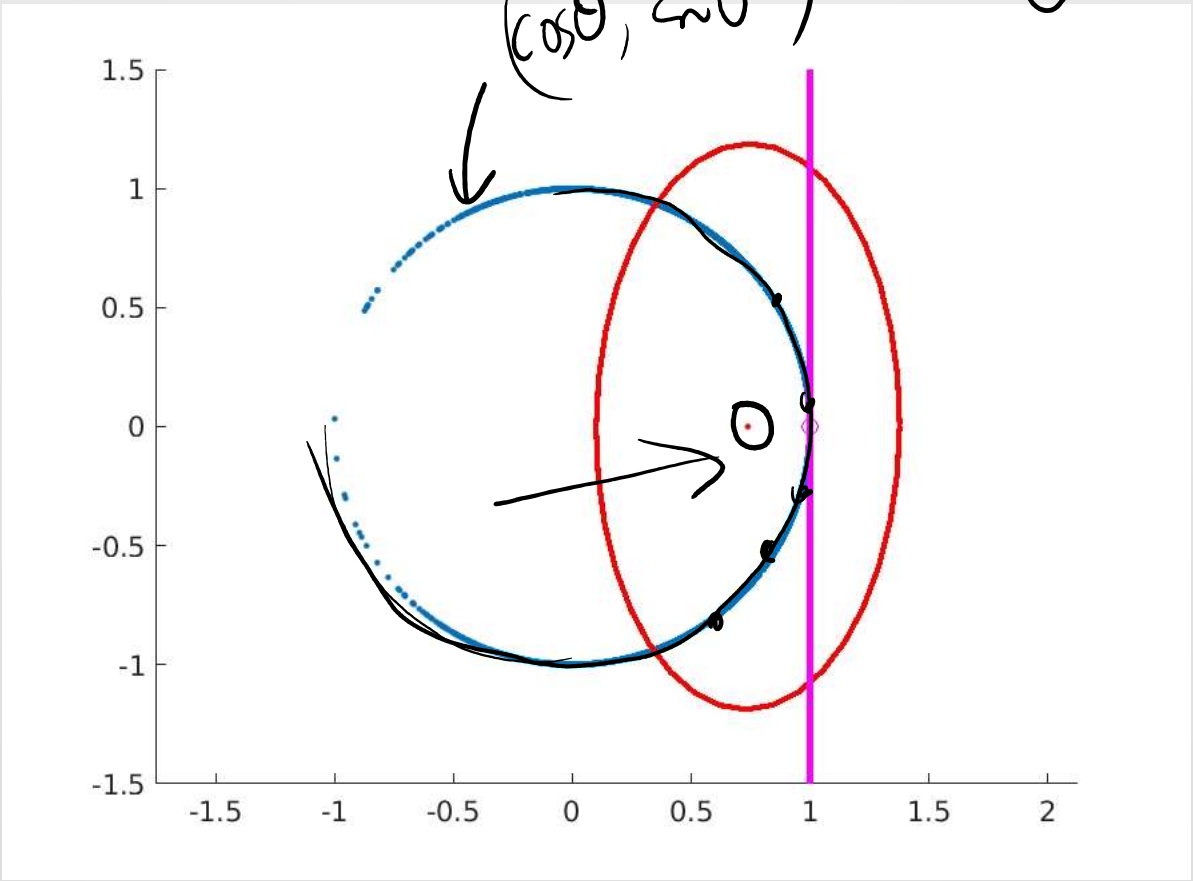
- The value is the same as

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} c f(\mathbf{x})$$

$$c > 0$$

$$c = \int f(z) \, dz \qquad c \geq 0$$

MAP Estimates Tend to be Sensible



Computing the MAP Estimate

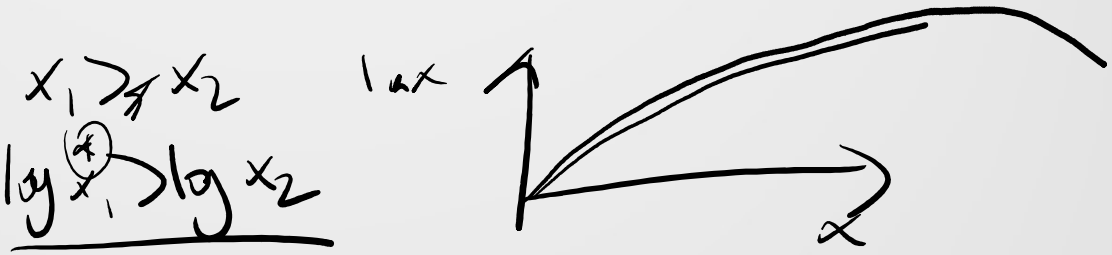
- We want to compute the maximum of the function

$$f'\left(\mathbf{x}_{1:k}|\mathbb{I}_k\right) = f\left(\mathbf{x}_0\right)\prod_{i=1}^{k} f\left(\mathbf{x}_i|\mathbf{x}_{i-1},\mathbf{u}_i\right) \times \prod_{i=1}^{k} L\left(\mathbf{x}_i;\mathbf{z}_i\right)$$

Computing the MAP Estimate

- Just directly optimizing this can lead to numerical over or underflow issues
- However, the form of this function is the product of terms, each of which is non-negative
- Therefore, we can take the log of the distribution

MAP and Logs



- The log of a function has the property that it is asymptotically increasing
- Therefore,

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} f'(\mathbf{x})$$

$$= \arg\max_{\mathbf{x}} \ln f'(\mathbf{x})$$

Computing the MAP Estimate

- Substituting,

$$\ln f'\left(\mathbf{x}_{0:k}|\mathbb{I}_k\right) = \ln f\left(\mathbf{x}_0\right) + \sum_{i=1}^{k} \ln f\left(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{u}_i\right) + \sum_{i=1}^{k} \ln L\left(\mathbf{x}_i; \mathbf{z}_i\right)$$

State Transition Probabilities and Likelihoods

- We now need to substitute for the expressions for the transition density and likelihood functions

$$f\left(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{u}_k\right) = f_{\mathbf{v}}\left(\mathbf{v}_k \stackrel{\swarrow}{=} \mathbf{e}\left[\mathbf{x}_k,\mathbf{x}_{k-1},\mathbf{u}_k\right]\right)$$

$$f\left(\mathbf{z}_k|\mathbf{x}_k\right) = f_{\mathbf{w}}\left(\mathbf{w}_k = \mathbf{l}\left[\mathbf{x}_k,\mathbf{z}_k\right]\right)$$

Linear Case

- For the sensor likelihood equations, these are

$$L(\mathbf{x}_k; \mathbf{z}_k) \propto \exp\left\{-\frac{1}{2}\mathbf{l}\left[\mathbf{x}_k, \mathbf{z}_k\right]^\top \mathbf{R}_k^{-1}\mathbf{l}\left[\mathbf{x}_k, \mathbf{z}_k\right]\right\}$$

- Taking logs,

$$\ln L(\mathbf{x}_k; \mathbf{z}_k) = -\frac{1}{2}\mathbf{l}\left[\mathbf{x}_k, \mathbf{z}_k\right]^\top \mathbf{R}_k^{-1}\mathbf{l}\left[\mathbf{x}_k, \mathbf{z}_k\right] + c$$

Linear Case

- For the state transition equations, we can substitute for the inverse process model to get

$$f\left(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{u}_k\right) \propto \exp\left\{-\frac{1}{2}\mathbf{e}\left[\mathbf{x}_k,\mathbf{x}_{k-1},\mathbf{u}_k\right]^\top \mathbf{Q}_k^{-1}\mathbf{e}\left[\mathbf{x}_k,\mathbf{x}_{k-1},\mathbf{u}_k\right]\right\}$$

$e = x_t - F x_{t-1}$

- Taking logs,

$$\ln f\left(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{u}_k\right) = -\frac{1}{2}\mathbf{e}\left[\mathbf{x}_k,\mathbf{x}_{k-1},\mathbf{u}_k\right]^\top \mathbf{Q}_k^{-1}\mathbf{e}\left[\mathbf{x}_k,\mathbf{x}_{k-1},\mathbf{u}_k\right] + d$$

Putting Everything Together

- Substituting Everything together, we get:

$$\ln f'\left(\mathbf{x}_{1:k}|\mathbb{I}_k\right) = -\frac{1}{2}\sum_{i=1}^{k}\mathbf{e}\left[\mathbf{x}_k,\mathbf{x}_{k-1},\mathbf{u}_k\right]^\top \mathbf{Q}_k^{-1}\mathbf{e}\left[\mathbf{x}_k,\mathbf{x}_{k-1},\mathbf{u}_k\right]$$

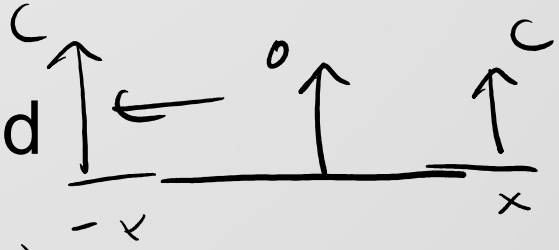
$$-\frac{1}{2}\sum_{i=1}^{k}\mathbf{l}\left[\mathbf{x}_k,\mathbf{z}_k\right]^\top \mathbf{R}_k^{-1}\mathbf{l}\left[\mathbf{x}_k,\mathbf{z}_k\right] + e$$

Need for Optimization

- We want to compute the solution

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \ln f'(\mathbf{x})$$

- However, the solution includes lots of –ve signs which is a bit inconvenient
- We can flip signs and scale to find



$$\mathbf{x}^* = \arg\min_{\mathbf{x}} -2\ln f'(\mathbf{x})$$

Putting Everything Together

- Therefore, the goal is to compute

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} c\left(\mathbf{x}_{1:k} | \mathbb{I}_k\right)$$

where

$$c\left(\mathbf{x}_{1:k} | \mathbb{I}_k\right) = \sum_{i=1}^{k} \mathbf{e}\left[\mathbf{x}_k, \mathbf{x}_{k-1}, \mathbf{u}_k\right]^\top \mathbf{Q}_k^{-1} \mathbf{e}\left[\mathbf{x}_k, \mathbf{x}_{k-1}, \mathbf{u}_k\right]$$

$$+ \sum_{i=1}^{k} \mathbf{l}\left[\mathbf{x}_k, \mathbf{z}_k\right]^\top \mathbf{R}_k^{-1} \mathbf{l}\left[\mathbf{x}_k, \mathbf{z}_k\right]$$

Optimization Algorithms

- *Motivation*
- *Interpreting the Graph*
- *Maximum Likelihood Estimation*
- Optimization
- *Approximate Covariance Calculations*

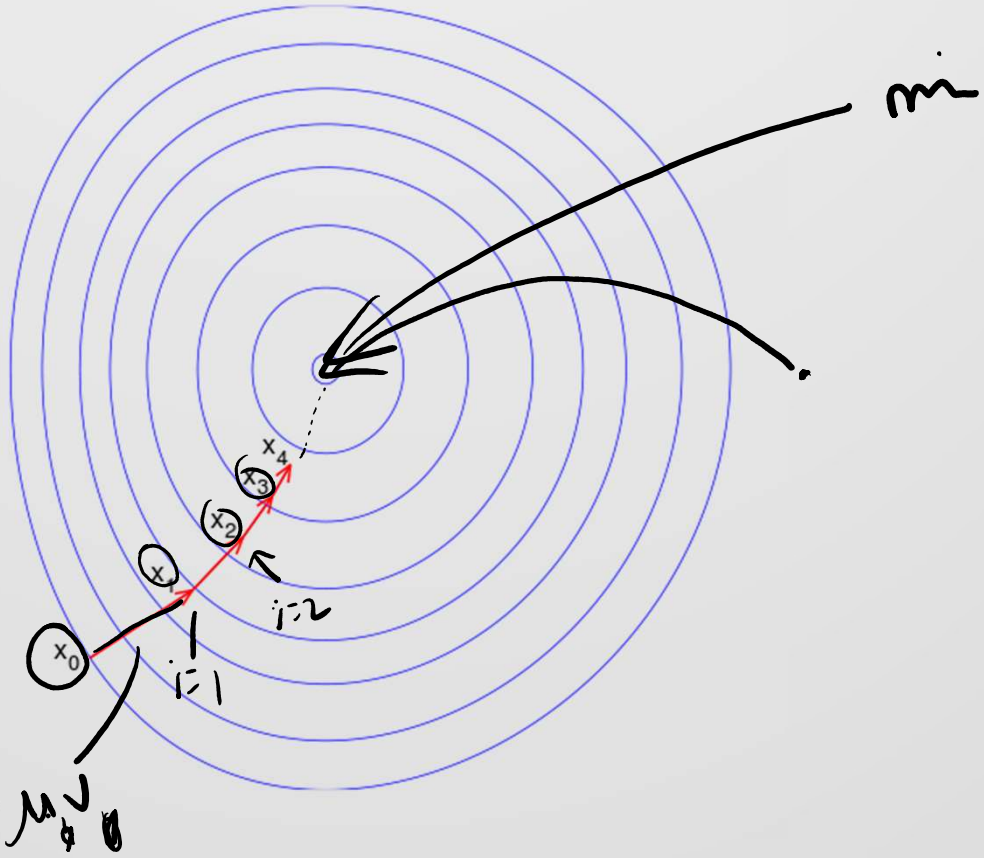
Need for Optimization

- The MAP estimate is computed from an optimization process
- If the system is completely linear, you can show that the MAP estimate can be solved directly using matrix inverses
- However, in general the system is not linear
- Therefore, we are solving a nonlinear least squares estimation

Nonlinear Least Squares Optimization

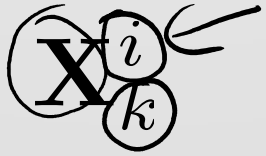
- Nonlinear systems rarely have a closed form solution
- Rather, we use an iterative approach which converges to the minimum

Iterative Nonlinear Optimization



Nonlinear Least Squares Optimization

- Nonlinear systems rarely have a closed form solution
- Rather, we use an iterative approach which converges to the minimum
- Since we clearly need more notation, we will write a solution in the optimizer as



Iterative Nonlinear Optimization

$\mathbf{X}_k^0 \leftarrow$ Initial value

$c^0 \leftarrow c\left(\mathbf{X}_k^0\right)$

$i \leftarrow 1$

repeat

$\quad \mathbf{v}_k^i \leftarrow$ Step direction

$\quad \mu_k^i \leftarrow$ Step length

$\quad \mathbf{X}_k^i = \mathbf{X}_k^{i-1} + \mu_k^i \mathbf{v}_k^i$

$\quad c^i \leftarrow c\left(\mathbf{X}_k^i\right)$

$\quad i \leftarrow i + 1$

until $|c^i - c^{i-1}| \leq \epsilon_c$ or $|\mathbf{X}_k^i - \mathbf{X}_k^{i-1}| \leq \epsilon_X$

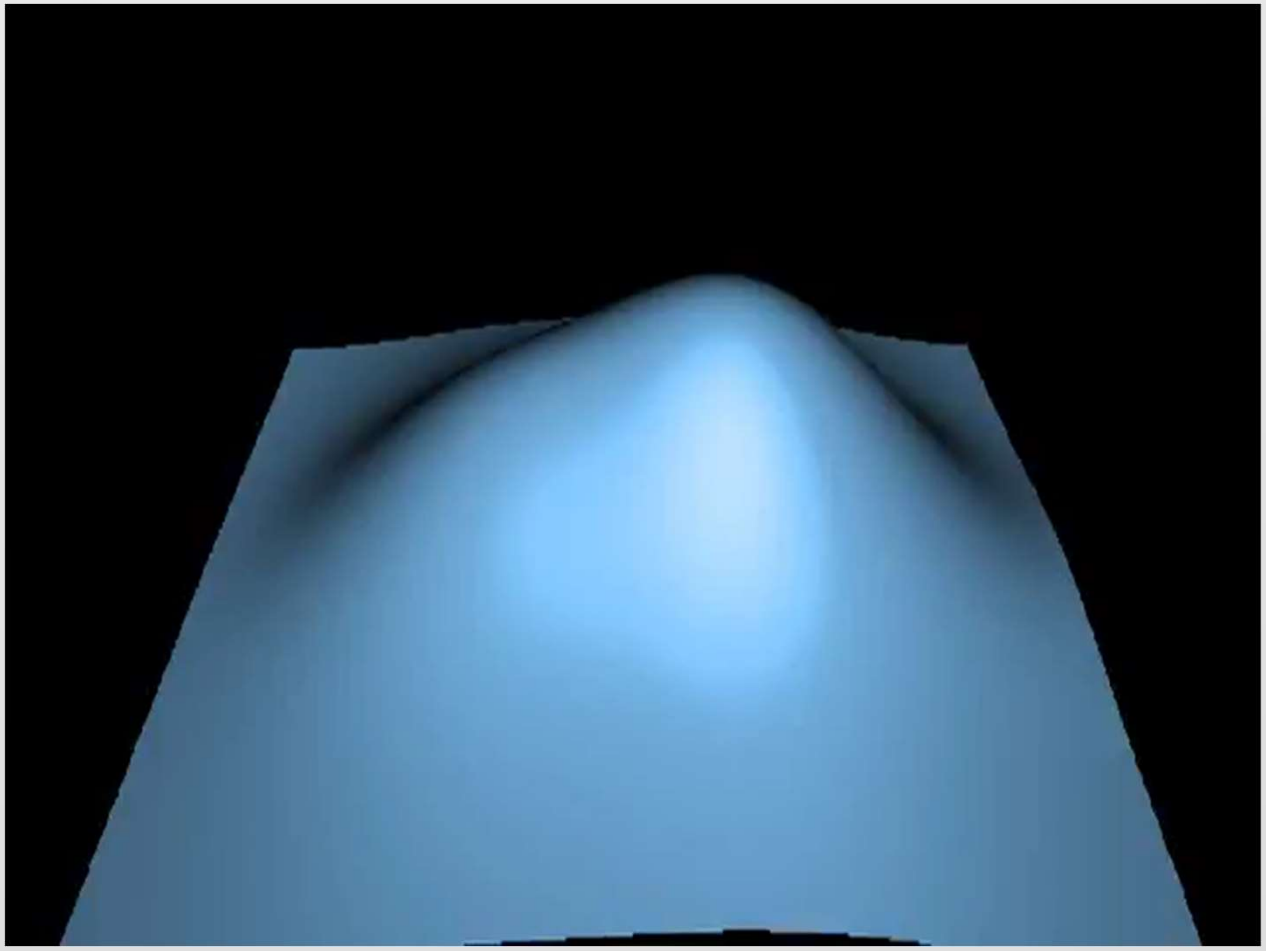
Gradient Descent Optimization

- Chooses the direction in which the cost function will fall the fastest
- The direction of greatest descent is the *negative* of the gradient of the cost function

$$\mathbf{X}_k^{i+1} = \mathbf{X}_k^i - \mathbf{J}\left(\mathbf{X}_k^i\right)$$

$$\mathbf{J} = \nabla c\left(\mathbf{X}_k^i\right) = \left[\frac{\partial c\left(\mathbf{X}_k^i\right)}{\partial x_1} \quad \dots \quad \frac{\partial c\left(\mathbf{X}_k^i\right)}{\partial x_n}\right]^\top$$

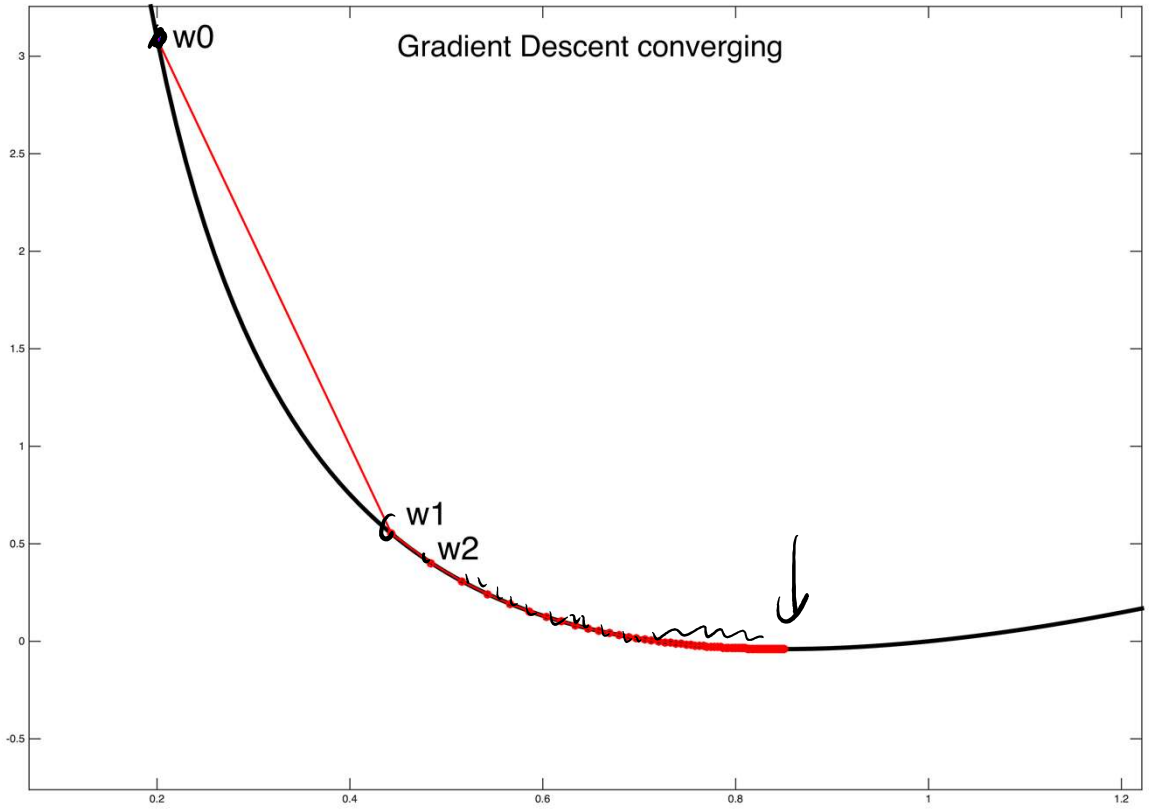
Gradient Descent (Ascent!) at Work



From https://www.youtube.com/watch?v=L2W5SfGu09M

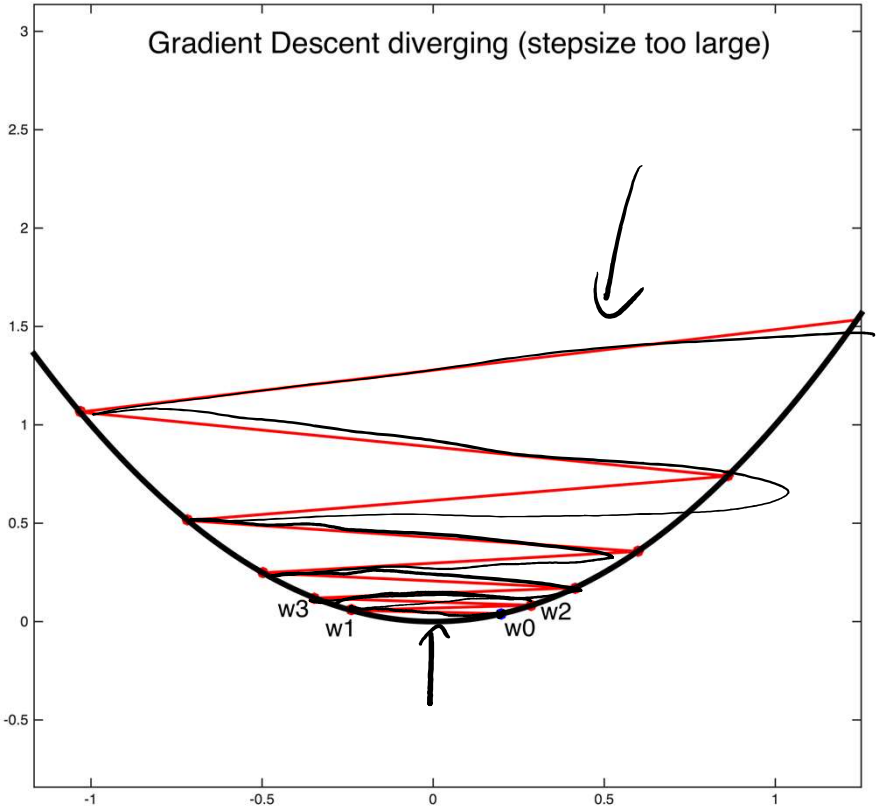
Effect of the Update

Good Step Size



From "Gradient Descent (and Beyond)"

Bad Step Size



$\mu = 1$

From "Gradient Descent (and Beyond)"

Scaling the Step Size

- We could try just changing the optimization direction to find somewhere "better"
- However most approaches assume that we simply made our step size "too big"
- Therefore, take the scaled step

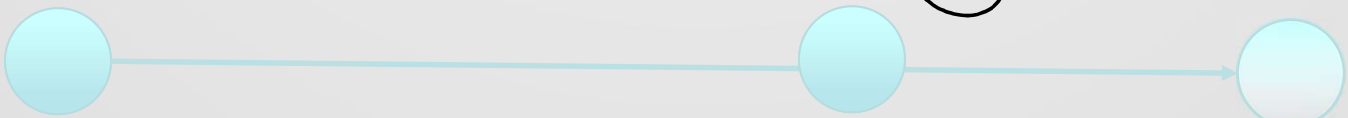
$$\mathbf{X}_k^{i+1} = \mathbf{X}_k^i - \mu_k^i \mathbf{J}\left(\mathbf{X}_k^i\right)$$

$$0 < \mu_k^i \leq 1$$

Scaled Step Size

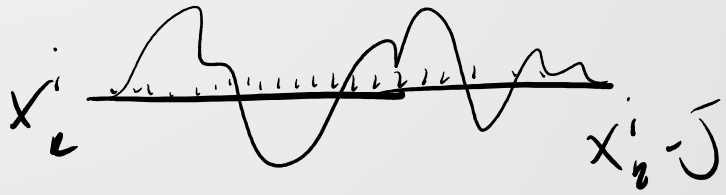
- We take a step along the gradient and change the step size

$$\mathbf{X}_k^i \qquad\qquad \mathbf{X}_k^i - \mu_k^i \mathbf{J} \; ?$$



$$c\left(\mathbf{X}_k^i\right) \qquad\qquad c\left(\mathbf{X}_k^i - \mu_k^i \mathbf{J}\right)$$

Choosing the Step Size



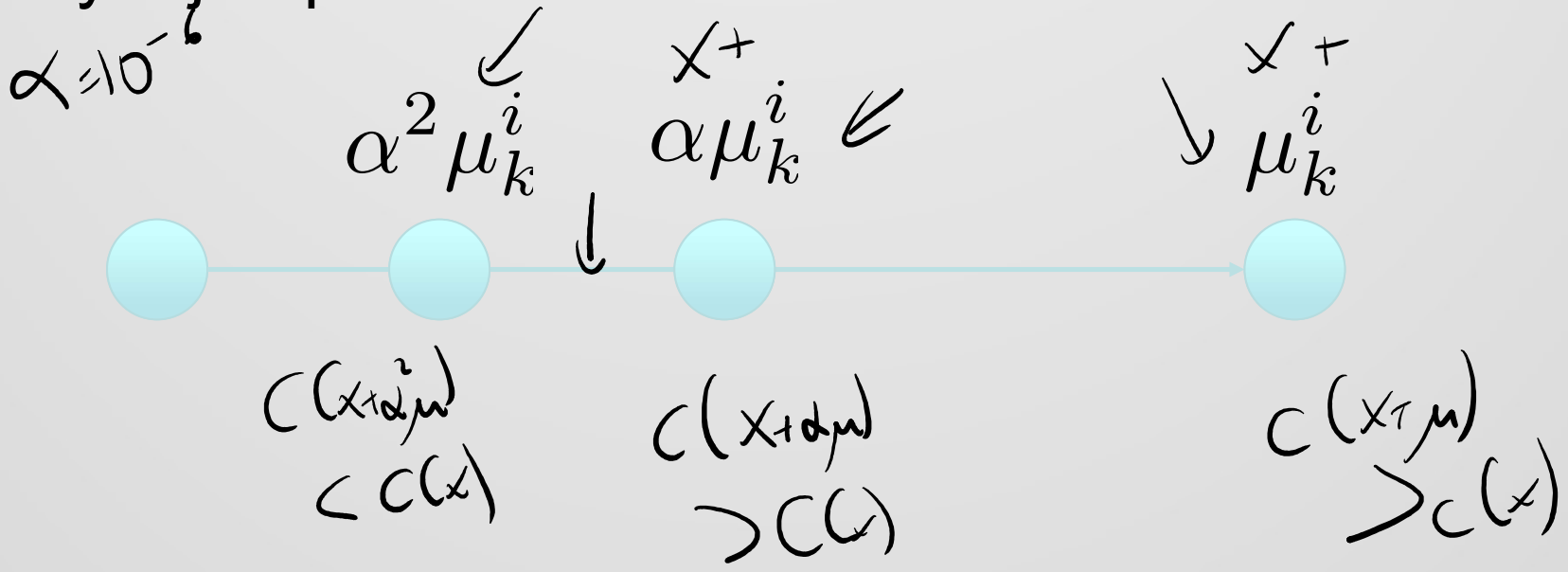
- The optimal solution is *exact line search*
- The step size is given by the solution

$$\arg\min_{\mu_k^i} c\left(\mathbf{X}_k^i - \mu_k^i \mathbf{J}\right)$$

- However, the actual cost function is completely arbitrary

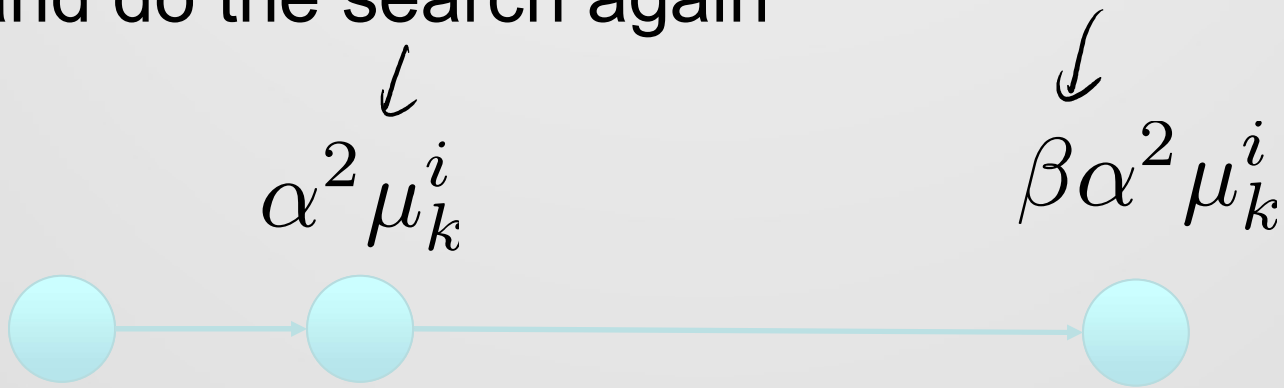
Choosing the Step Size

- Most versions use some kind of *backtracking line search*
- Try a jump and scale back on it if it's bad



Choosing the Step Size

- In the next step, scale the step size up by a factor β and do the search again

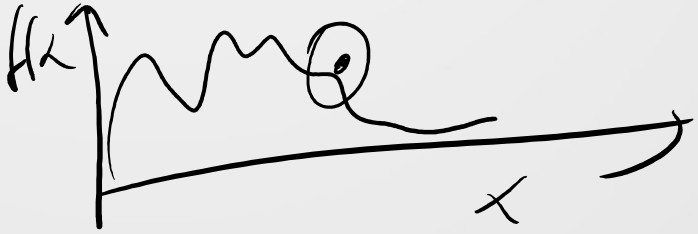


$$\beta \alpha^2 < 1$$

Newton-Raphson

- Gradient descent can find the local minimum
- However, its convergence can be very slow
- There are lots of variants of nonlinear optimization algorithms which have been derived
- Here we'll look at Newton, Gauss-Newton and Levenberg-Marquardt

Newton's Method



- The idea is to approximate the function by a quadratric and pick the step to optimize it
- The quadratic approximation is obtained from Taylor's series,

$$f(x + \delta x) \approx f(x) + f'(x)\delta x + \frac{1}{2} f''(x)\delta x^2$$

Newton's Method

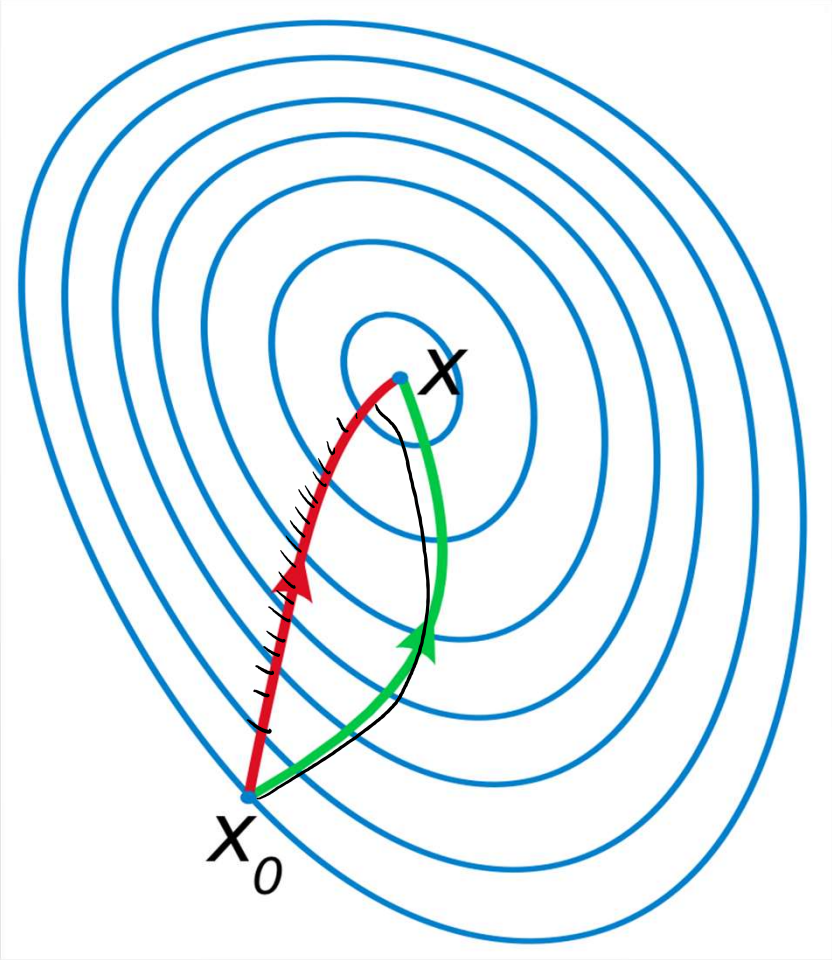
- Taking derivatives with respect to the step size,

$$\frac{\mathrm{d}f(x+\delta x)}{\mathrm{d}\delta x} = f'(x) + f''(x)\delta x$$

- Since the left hand side is 0,

$$\delta x = -\frac{f'(x)}{f''(x)}$$

Trajectory Taken by Newton's Method



From Newton's method in optimization

Newton's Method

- The multi-dimensional version is

$$\mathbf{X}_k^{i+1} = \mathbf{X}_k^i - \gamma \left[\mathbf{H}c\left(\mathbf{X}_k^i\right)\right]^{-1} \nabla c\left(\mathbf{X}_k^i\right)$$

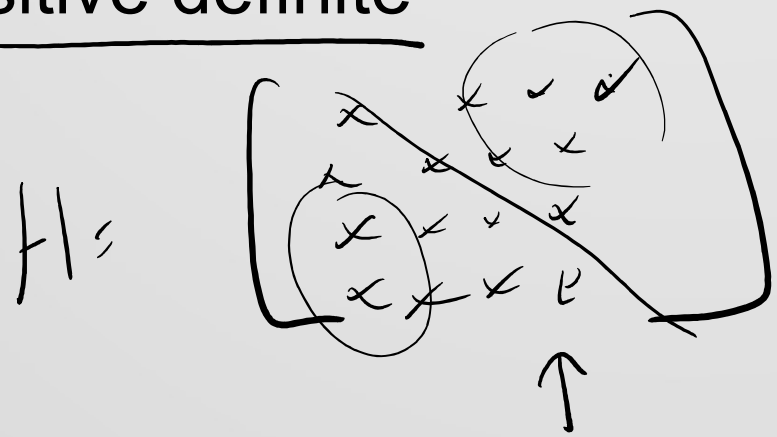
where

$$\mathbf{H}c\left(\mathbf{X}_k^i\right) = \begin{bmatrix} \frac{\partial^2 c\left(\mathbf{X}_k^i\right)}{\partial x_1^2} & \cdots & \frac{\partial^2 c\left(\mathbf{X}_k^i\right)}{\partial x_1 x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 c\left(\mathbf{X}_k^i\right)}{\partial x_n x_1} & \cdots & \frac{\partial^2 c\left(\mathbf{X}_k^i\right)}{\partial x_n^2} \end{bmatrix}^\top$$

Issues with Newton's Method

- It requires the calculation of the Hessian which can be *very expensive* to compute

- The method blows up if the Hessian is non positive definite



$eig(H) = \lambda_1, \ldots, \lambda_n$

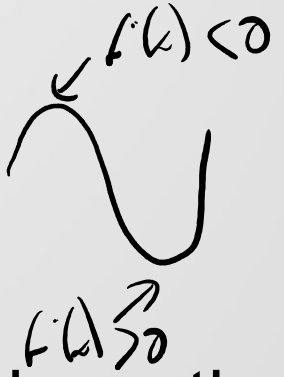
$\lambda_i > 0 \quad i = 1, \ldots n$

$\lambda_i <$

Non-Positiveness with the Hessian

- The easiest way to see this is with the scalar case
- We computed the turning point value

$$\delta x = -\frac{f'(x)}{f''(x)}$$



- However, the type of turning point depends on the second derivative

$$f''(x)$$

- If this is negative we have found a local maximum!

Gauss-Newton

- This uses an approximate value for the Hessian,

$$\mathbf{X}_k^{i+1} = \mathbf{X}_k^i - \gamma \left(\boldsymbol{\nabla} c\left(\mathbf{X}_k^i\right)^\top \boldsymbol{\nabla} c\left(\mathbf{X}_k^i\right) \right)^{-1} \boldsymbol{\nabla} c\left(\mathbf{X}_k^i\right)$$

- The approximate value is the “square” of the Jacobian
- You don’t have to compute second derivatives
- This is guaranteed to be positive semidefininte

Limitations with Gauss-Newton

- Gauss Newton is not guaranteed to find a local minimum
- We also have the problem that if the Jacobian is nearly singular, the step might be very inaccurately calculated, if it can be calculated at all
- Levenberg-Marquardt attempts to combine the "slow and safe" approach from Gradient Descent with "faster and less safe" ness from Gauss Newton

Levenberg-Marquardt

- The method introduces a damping factor:

$$\mathbf{X}_k^{i+1} = \mathbf{X}_k^i - \gamma \left(\nabla c \left(\mathbf{X}_k^i \right)^\top \nabla c \left(\mathbf{X}_k^i \right) + \lambda \mathbf{I} \right)^{-1} \nabla c \left(\mathbf{X}_k^i \right)$$

$\lambda \approx 0$ $\quad$ $\lambda \gg 0$

- The damping factor is chosen dynamically, like the step size, to give the best result

Levenberg-Marquardt Pseudo-Code

$\lambda \leftarrow 10^{-3}$;
Repeat
1. Compute the gradient vector $\mathbf{b}$ and the Hessian matrix $\mathbf{H}$;
2. Compute the increment $\widehat{\delta\theta} = -(\mathbf{H} + \lambda I)^{-1}\mathbf{b}$;
3. If $f(\theta + \widehat{\delta\theta}) < f(\theta)$
   - $\lambda \leftarrow \frac{\lambda}{10}$;

   Else
   - Repeat
     - $\lambda \leftarrow \lambda \times 10$;
     - Compute the increment $\widehat{\delta\theta} = -(\mathbf{H} + \lambda I)^{-1}\mathbf{b}$;

     Until $f(\theta + \widehat{\delta\theta}) < f(\theta)$;

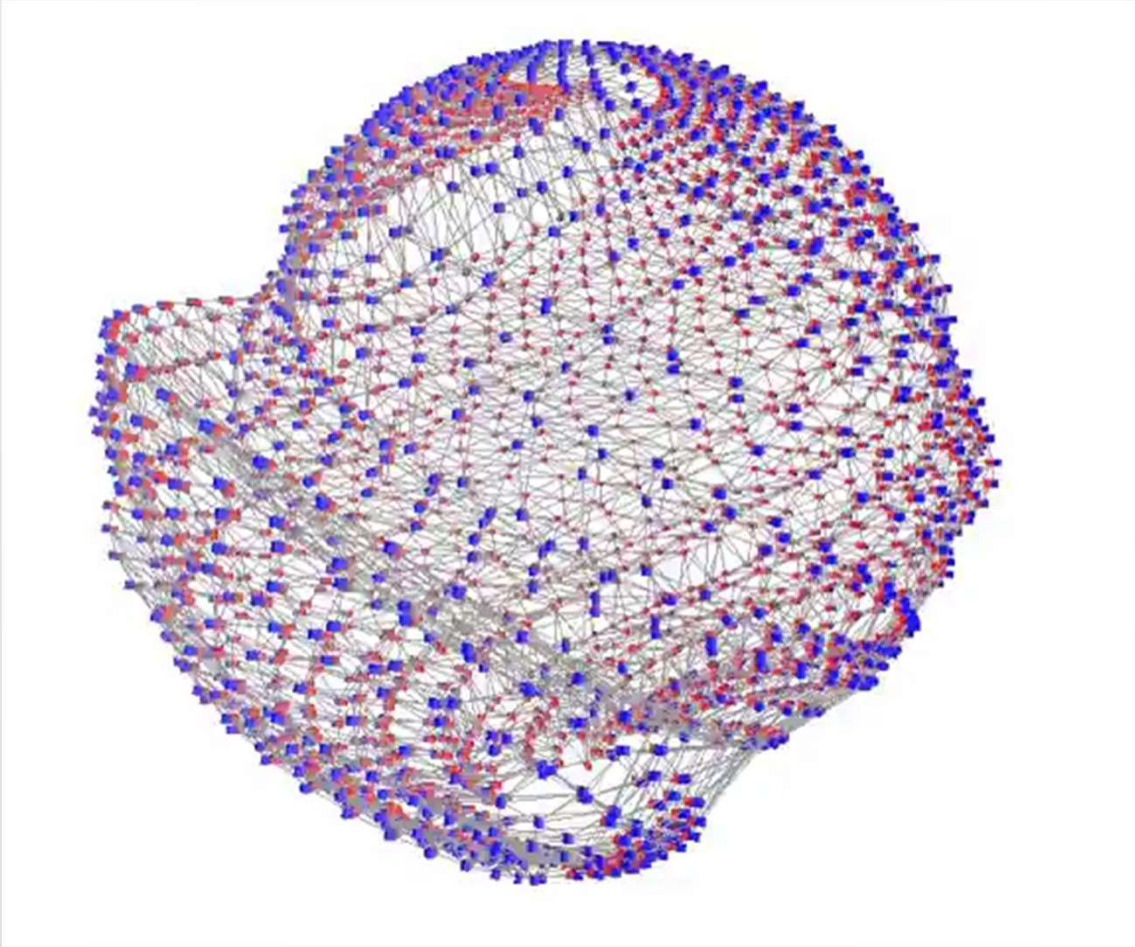
   Endif
4. $\theta \leftarrow \theta + \widehat{\delta\theta}$;

Until convergence.

From "HESSIAN MATRIX VS. GAUSS—NEWTON HESSIAN MATRIX", P. Chen, 2011

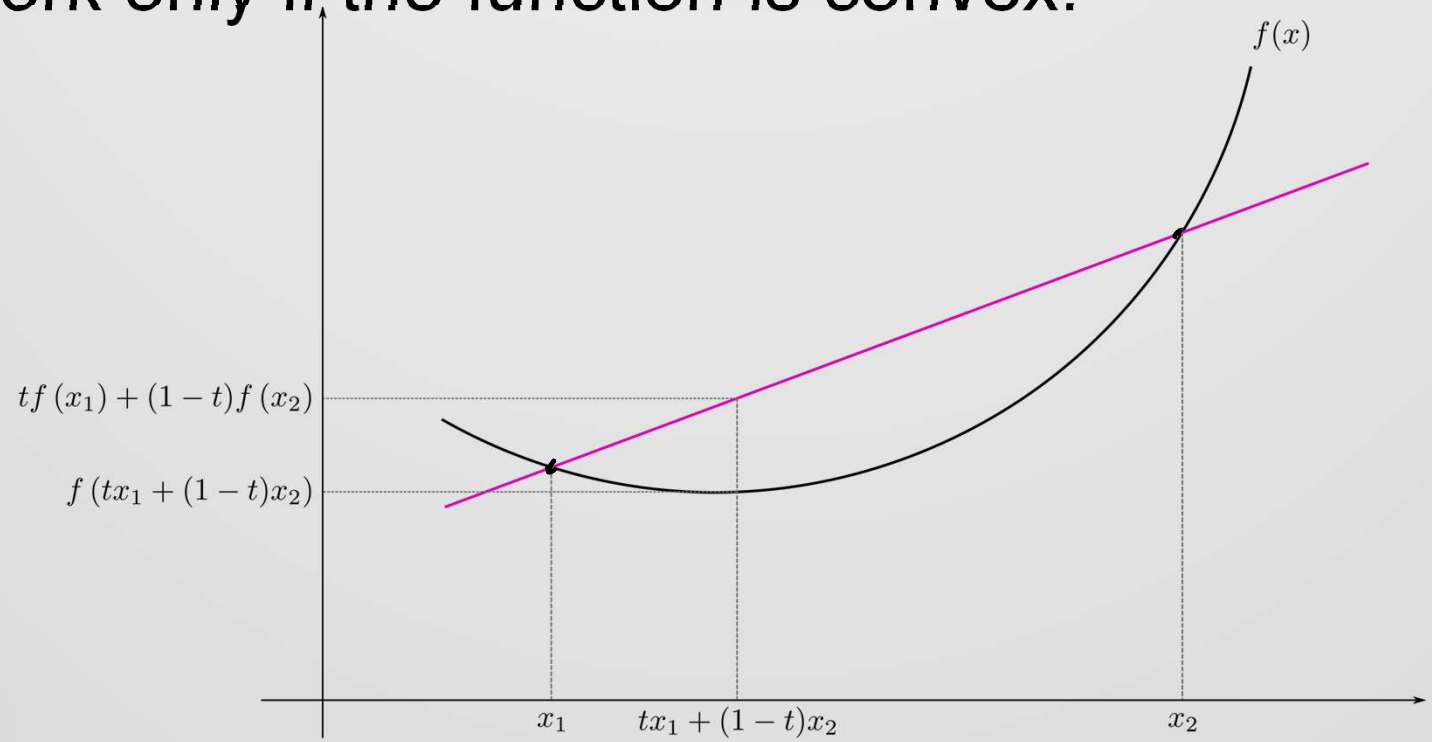
Challenge Optimization Problem

TORO



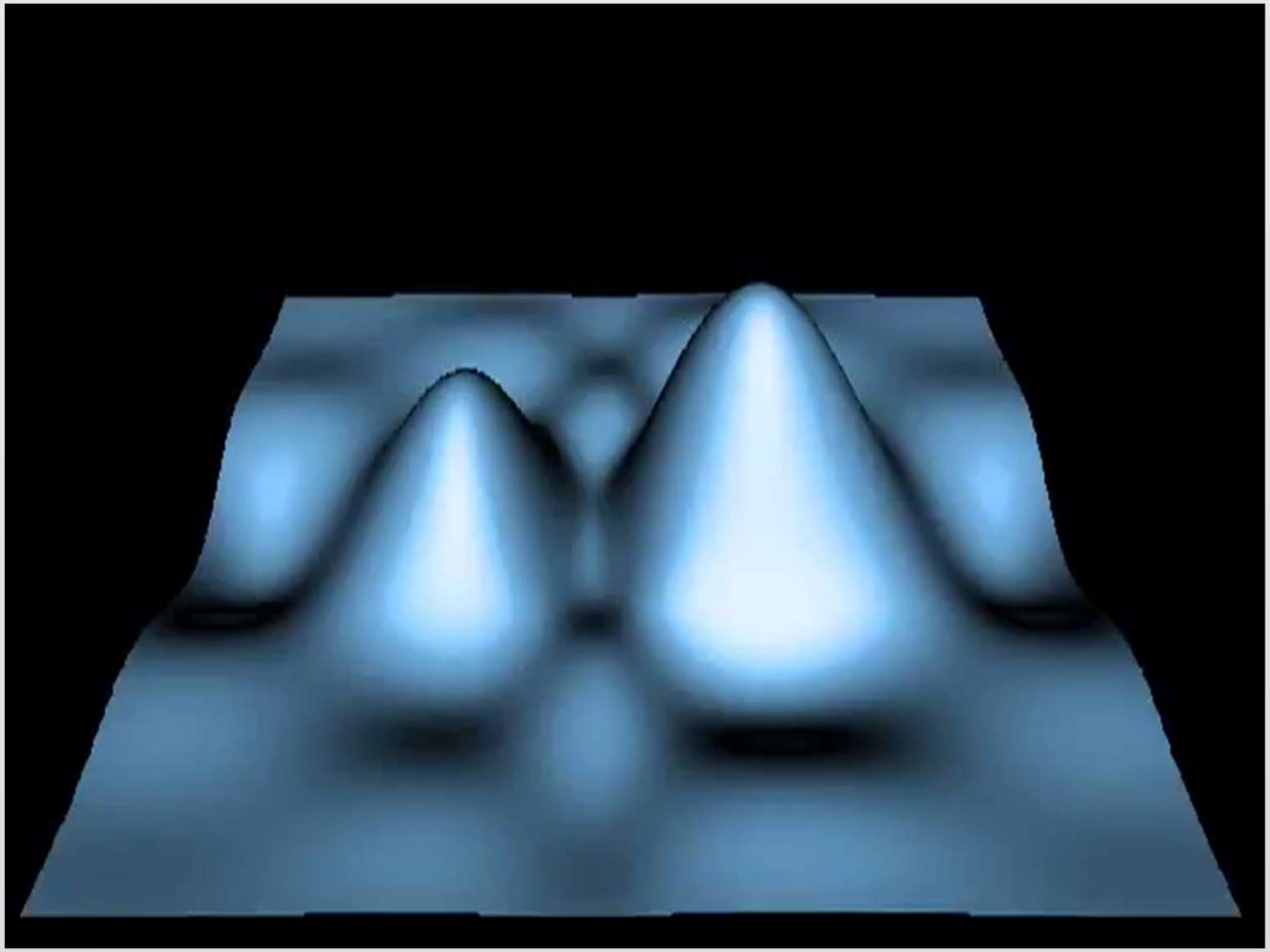
Convexity and Optimization

- Gradient descent approaches are guaranteed to work only if the function is convex:



From https://en.wikipedia.org/wiki/Convex_function

Gradient Ascent and non-Convex Functions



Addressing Issues of Non-Convexity

- Almost any real system is non-convex
- However, they have a “basin of attraction” for the solution
- The rule-of-thumb is to try to compute an initial solution which is as close as possible to the optimum
- There are new “certifiable” SLAM algorithms, but the maths is rather challenging

Approximate Covariance Calculations

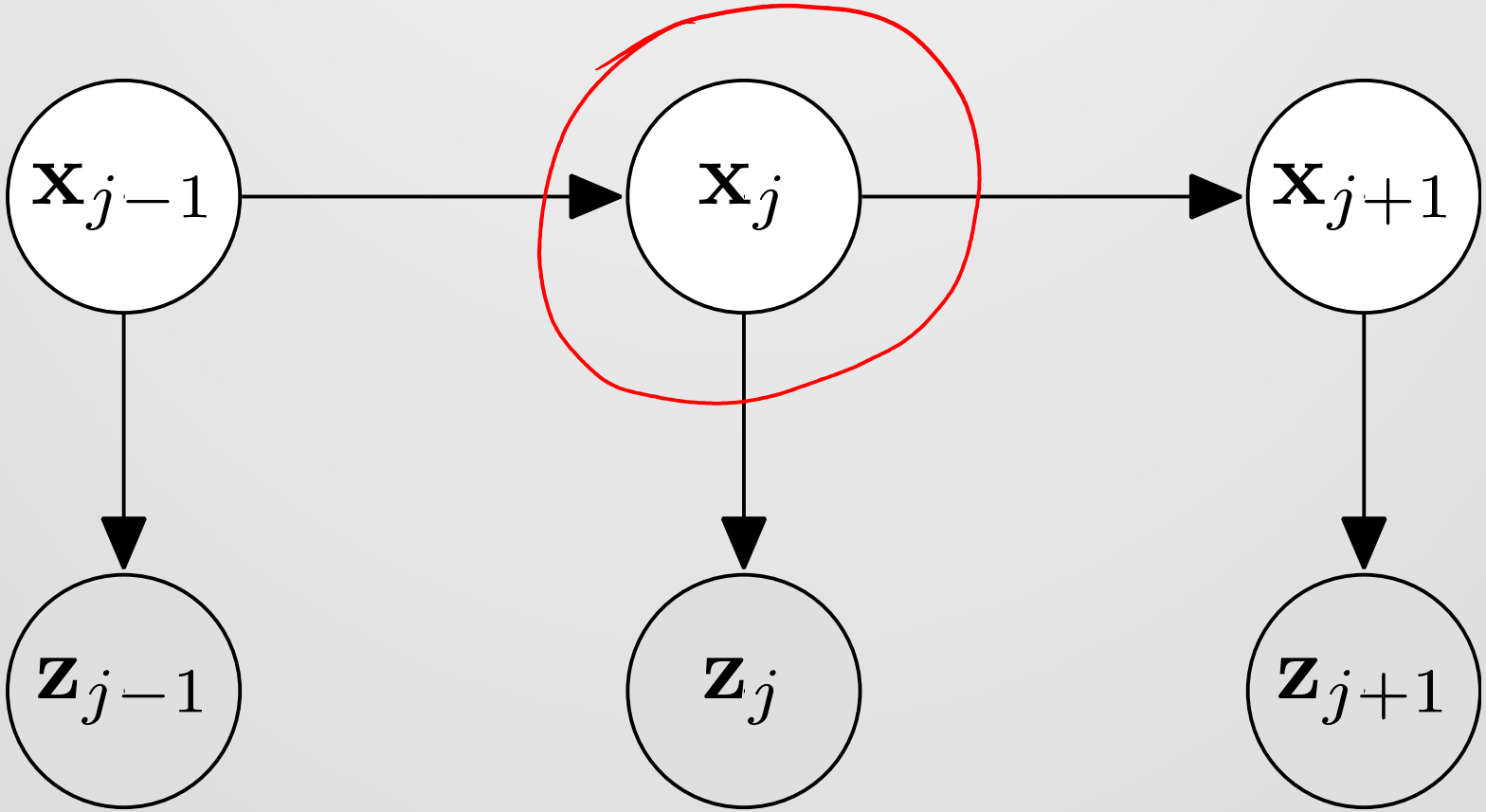
- *Motivation*
- *Interpreting the Graph*
- *Maximum Likelihood Estimation*
- *Optimization Algorithms*
- Approximate Covariance Calculations

Computing the Covariance

$$\mathbb{E}\left[(x_i - \hat{x}_i)^2\right]$$

- The covariance of a state estimate is *never used* when computing the MAP estimate
- However, in many cases we would like to estimate the covariance of the state estimate to determine things such as whether we have collected enough measurements, or to perform data association
- Computing it properly is challenging because we'd like to compute the expectation of a squared function

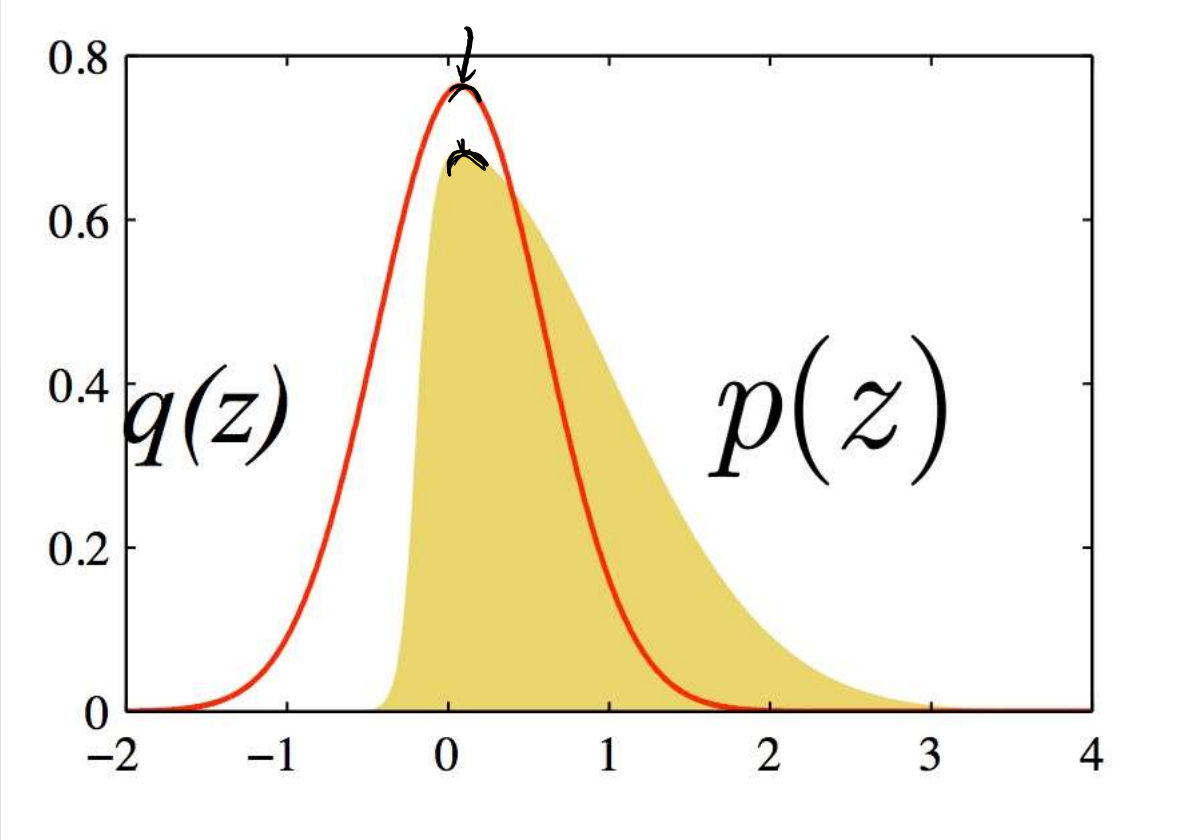
Computing the Covariance



Laplace Approximation

- The distribution around the MAP estimate is approximated by a Gaussian
- The mean of the Gaussian is the MAP estimate
- The covariance of the Gaussian is given by the local curvature of the distribution

The Laplace Approximation



The Laplace Approximation in a Graph

- For the graph, it turns out the curvature is actually given by the inverse of the Hessian
- Using various approximations, its value is given by

$$\left(\nabla c\left(\mathbf{X}_k^*\right)^\top \nabla c\left(\mathbf{X}_k^*\right) \right)^{-1}$$

marginals

$(\hat{x}, \rho)$

λI

- This value was actually worked out as part of the Levenberg-Marquardt update step

Summary

- To extract a point estimate from the graph, we use maximum aposterior (MAP) estimates
- These do not require calculation of the normalization constant and tend to put the estimate in the ~~correct~~ sensible place
- Computing the MAP estimate is a weighted nonlinear least squares estimation problem
- Recursive solutions exist to compute it

Bundle adjustment