graphics and vision gravis

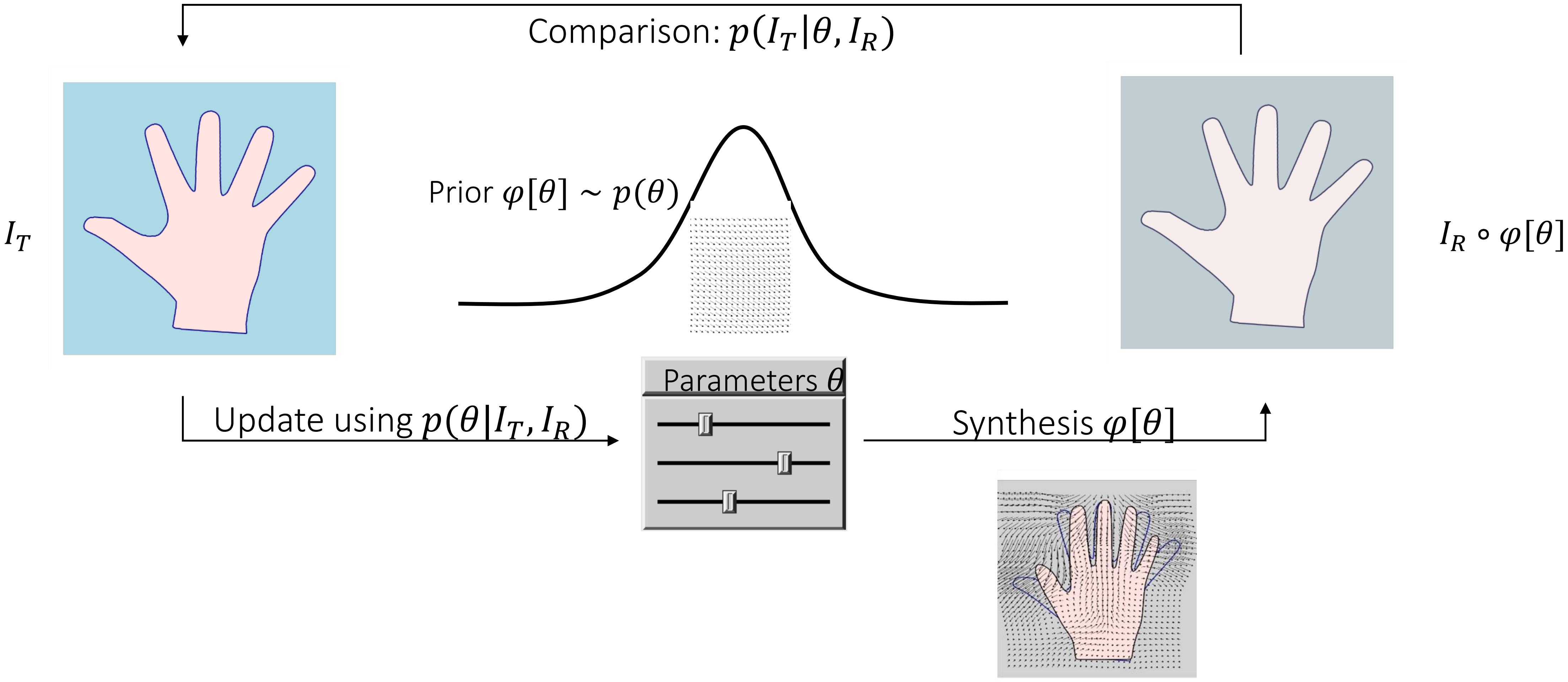
University of Basel

Probabilistic model fitting

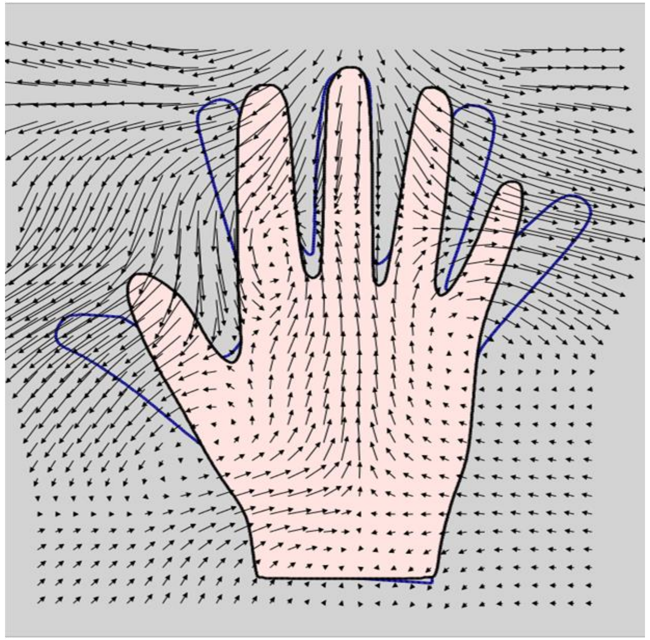
Marcel Lüthi

Graphics and Vision Research Group
Department of Mathematics and Computer Science
University of Basel

Reminder: Registration as analysis by synthesis



Reminder: Priors



Gaussian process

$$u \sim GP(\mu, k)$$

Represented using first r components

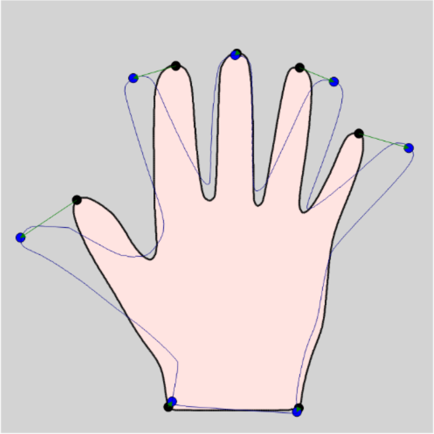
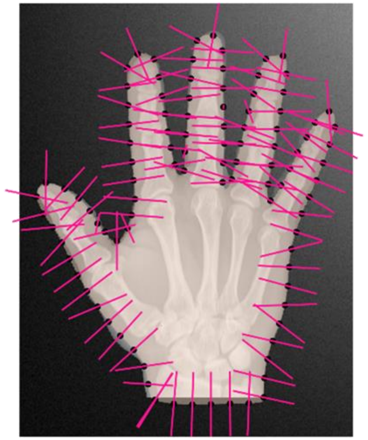
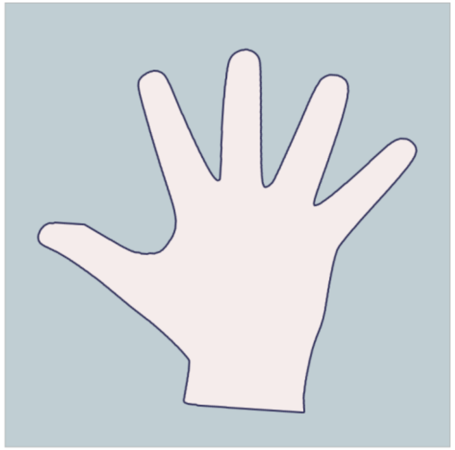
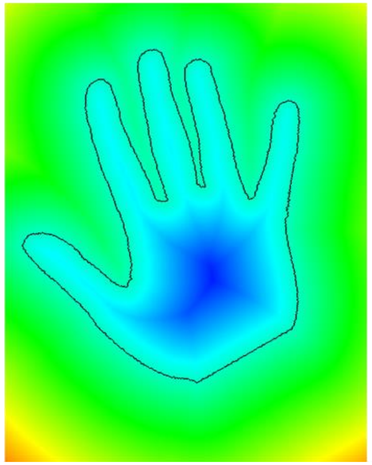
$$u = \mu + \sum_{i=1}^{r} \alpha_i \sqrt{\lambda_i}\, \phi_i, \qquad \alpha_i \sim N(0, 1)$$

Different GP-s lead to very different deformation models

- *All of them are parametric* $u \sim p(\theta)$.

Reminder: Likelihood functions

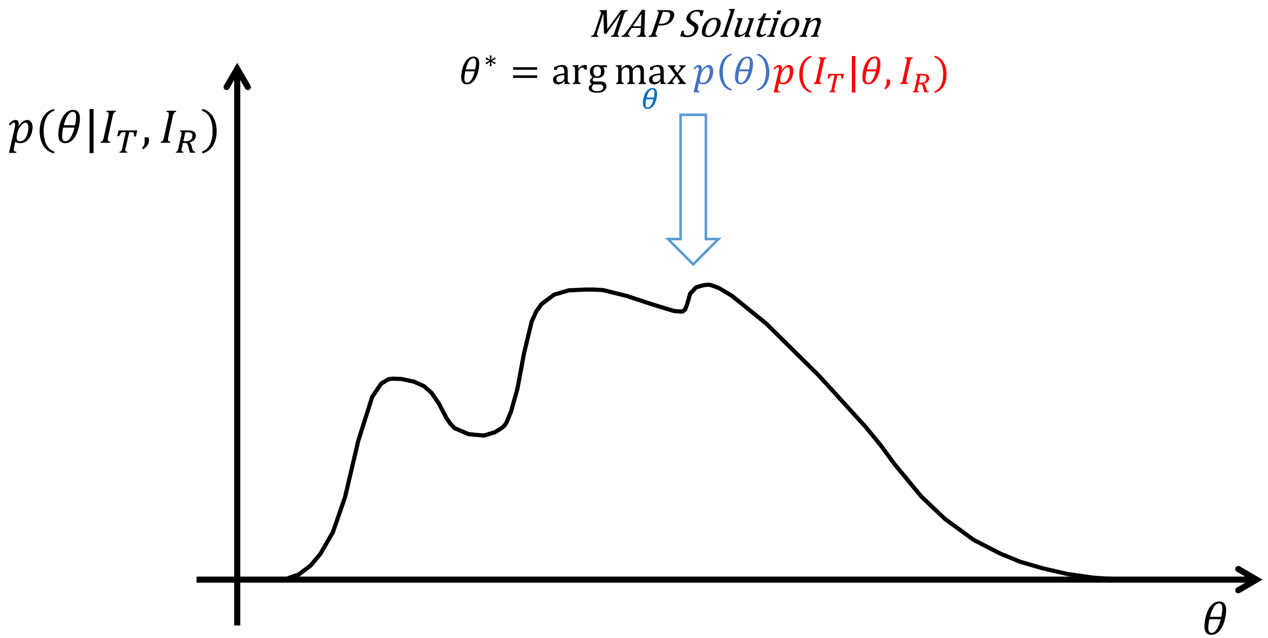
Likelihood function: $p(I_T|\theta, I_R)$

				
Comparison	Landmarks / landmark	Surface / image	Image / image	Surface / surface
Stochastic component	Noise on landmark points	Intensity profiles at surface boundary	Deviation of image intensity from reference image	(Distance to) exact position of surface

Reminder: Obtaining the posterior parameters

MAP-Estimate

$$\theta^* = \arg\max_{\theta} p(\theta|I_T, I_R) = \arg\max_{\theta} p(\theta)p(I_T|\theta, I_R)$$



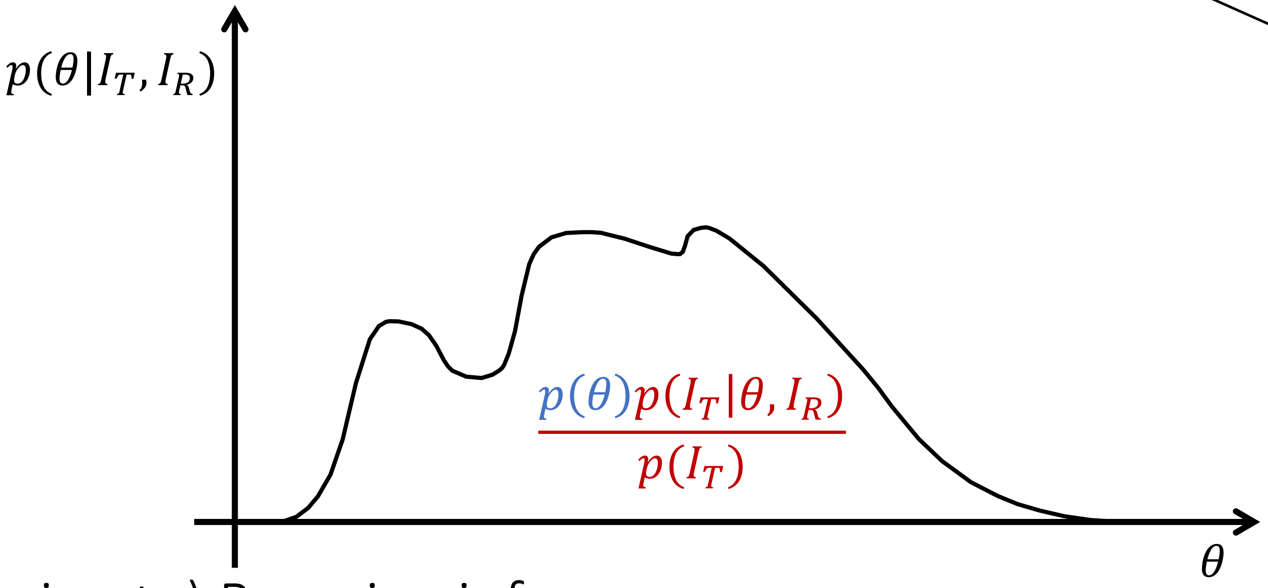
- Solving an optimization problem

Todays Lecture: Obtaining the posterior distribution

Full posterior distribution

$$p(\theta|I_T, I_R) = \frac{p(\theta)p(I_T|\theta, I_R)}{p(I_T)}$$

Infeasible to compute:
p(I_T)= $\int p(\theta)p(I_T|\theta\,)d\theta$



- Doing (approximate) Bayesian inference

Outline

- Basic idea: Sampling methods and MCMC
- The Metropolis-Hastings algorithm
  - The Metropolis algorithm
  - Implementing the Metropolis algorithm
  - The Metropolis-Hastings algorithm
- Example: 3D Landmark fitting

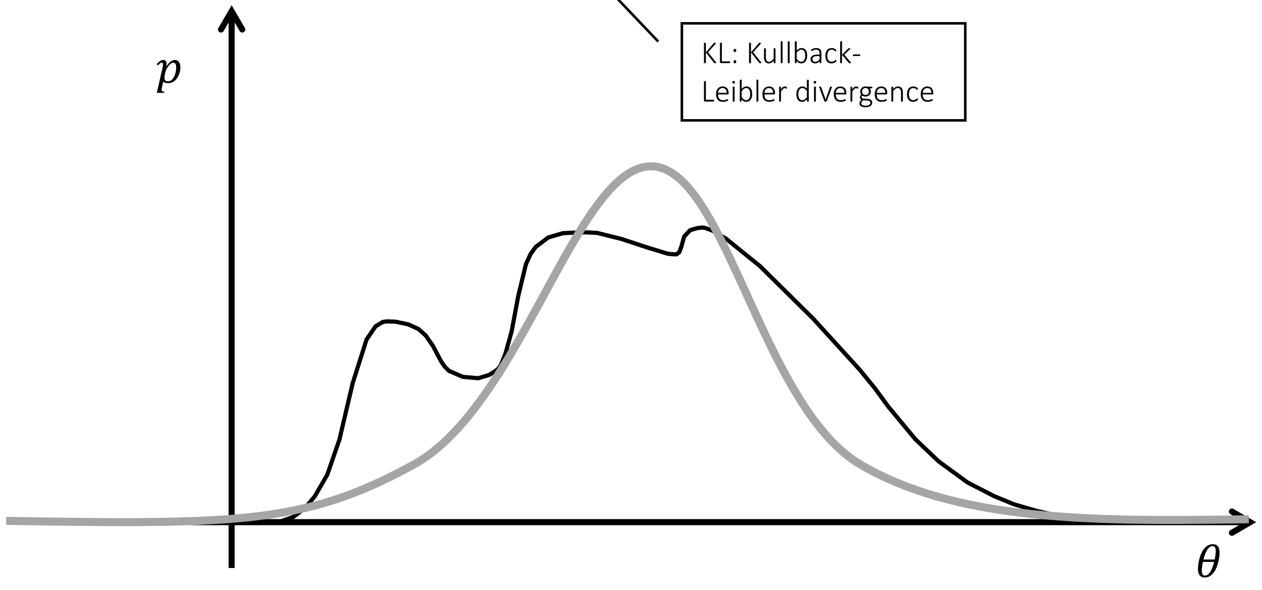
Approximate Bayesian Inference

Variational methods

- Function approximation $q(\theta)$

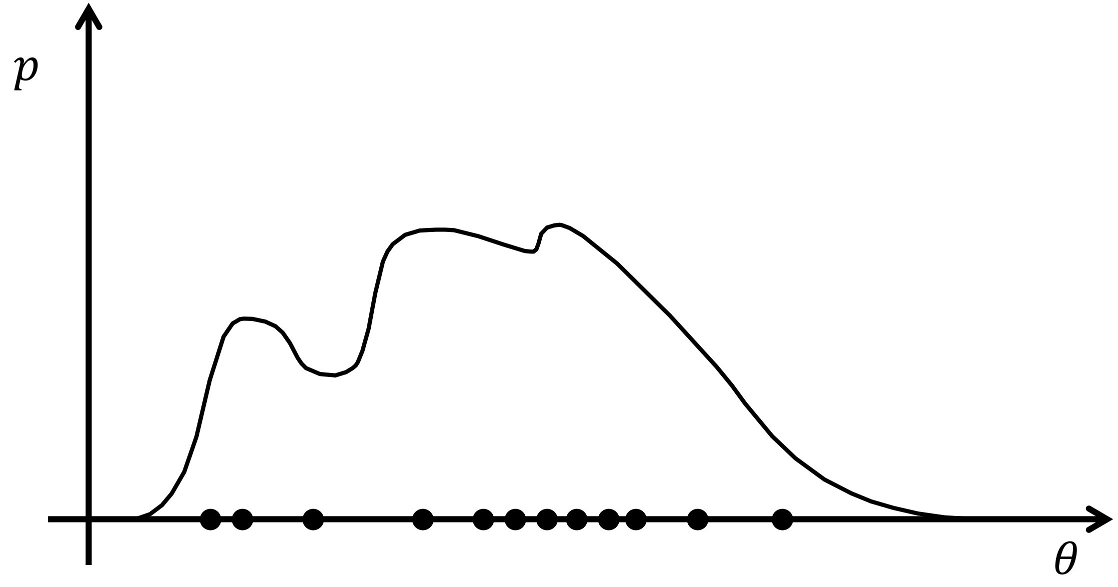
$$\arg\max_{q} \mathrm{KL}(q(\theta)|p(\theta|D))$$

KL: Kullback-Leibler divergence



Sampling methods

- Numeric approximations through simulation



Sampling Methods

- Simulate a distribution p through random samples x_i
- Evaluate expectation (of some function f of random variable X)

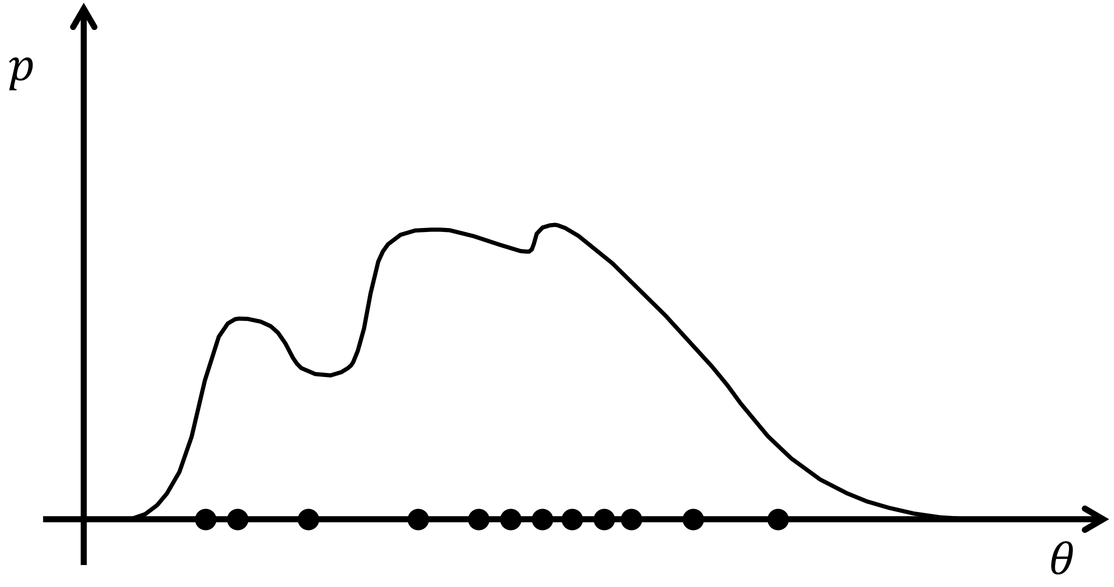
$$E[f(X)] = \int f(x)p(x)dx$$

$$E[f(X)] \approx \hat{f} = \frac{1}{N}\sum_i^N f(x_i), \qquad x_i \sim p(x)$$

This is difficult!

$$V[\hat{f}(X)] \sim O\left(\frac{1}{N}\right)$$

- *"Independent" of dimensionality of X*
- *More samples increase accuracy*



Sampling from a Distribution

- Easy for standard distributions … is it?
  - Uniform
  - Gaussian

```
Random.nextDouble()
Random.nextGaussian()
```

- How to sample from more complex distributions?
  - Beta, Exponential, Chi square, Gamma, …
  - Posteriors are very often not in a “nice” standard text book form

- *We need to sample from an unknown posterior with only unnormalized, expensive point-wise evaluation* ☹

Markov Chain Monte Carlo

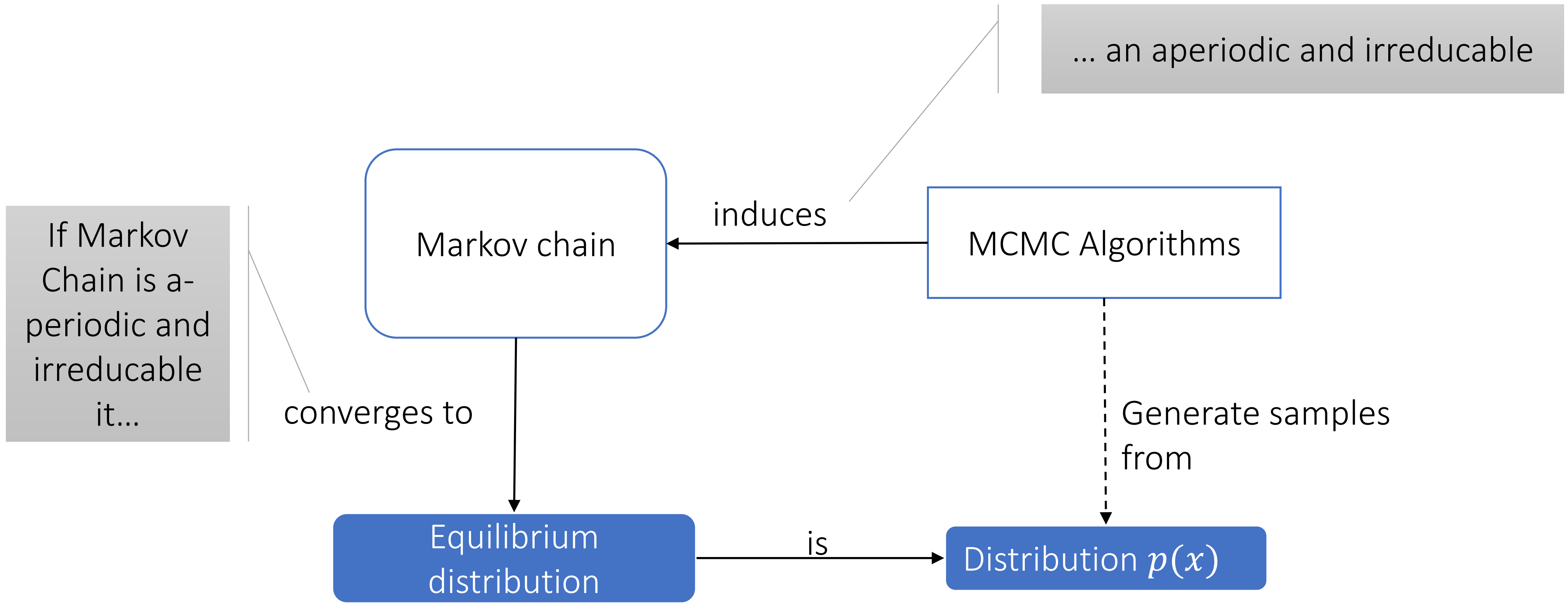
Markov Chain Monte Carlo Methods (MCMC)

Idea: Design a *Markov Chain* such that samples x obey the target distribution p

Concept: *“Use an already existing sample to produce the next one”*

- Many successful practical applications
  - Proven: developed in the 1950/1970ies (Metropolis/Hastings)
- Direct mapping of computing power to approximation accuracy

MCMC: An ingenious mathematical construction



No need to understand this now: more details follow!

The Metropolis Algorithm

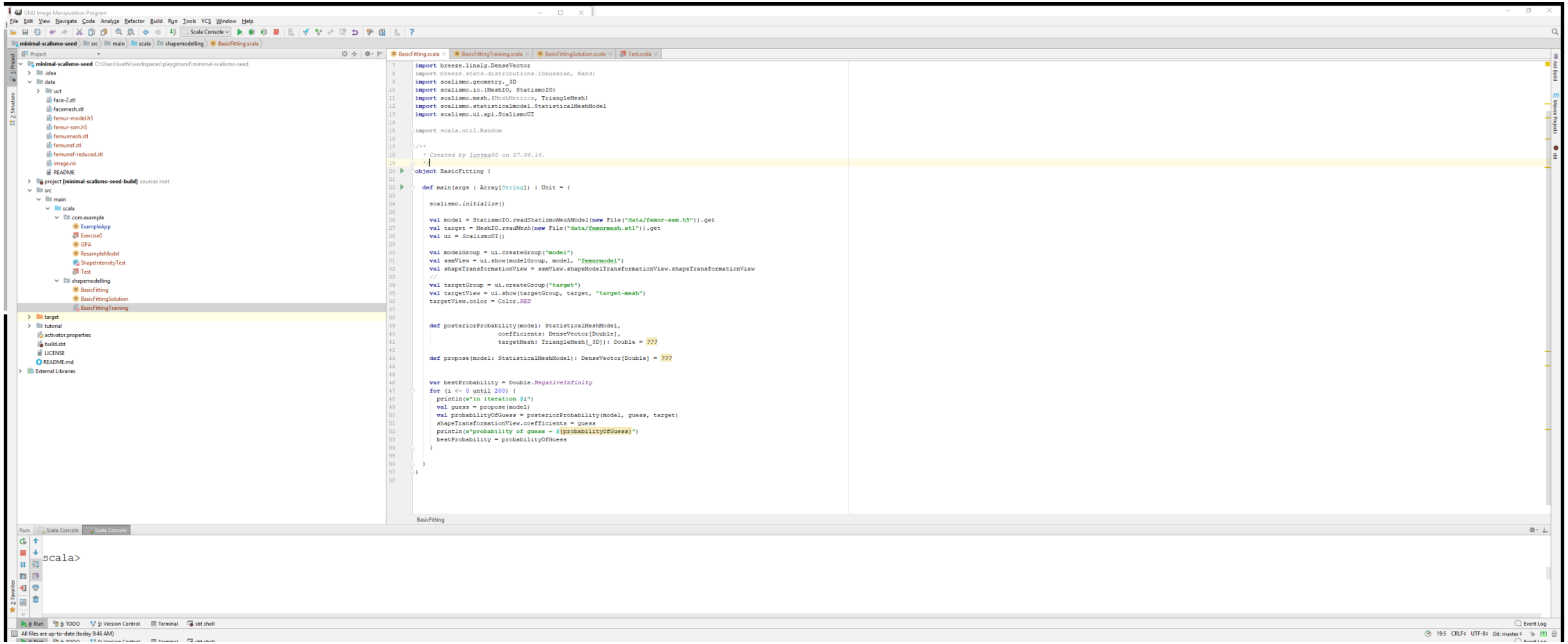
Requirements:

- Proposal distribution $Q(\boldsymbol{x}'|\boldsymbol{x})$ – *must generate samples, symmetric*
- Target distribution $P(\boldsymbol{x})$ – *with point-wise evaluation*

Result:

- Stream of samples approximately from $P(\boldsymbol{x})$

- Initialize with sample $\boldsymbol{x}$
- Generate next sample, with current sample $\boldsymbol{x}$
  1. Draw a sample $\boldsymbol{x}'$ from $Q(\boldsymbol{x}'|\boldsymbol{x})$ (“proposal”)
  2. With *probability* $\alpha = \min\left\{\frac{P(\boldsymbol{x}')}{P(\boldsymbol{x})}, 1\right\}$ accept $\boldsymbol{x}'$ as new state $\boldsymbol{x}$
  3. Emit current state $\boldsymbol{x}$ as sample


BasicFitting.scala
BasicFittingTraining.scala
BasicFittingSolution.scala
Test.scala
import breeze.linalg.DenseVector
import breeze.stats.distributions.{Gaussian, Rand}
import scalismo.geometry._3D
import scalismo.io.{MeshIO, StatismoIO}
import scalismo.mesh.{MeshMetrics, TriangleMesh}
import scalismo.statisticalmodel.StatisticalMeshModel
import scalismo.ui.api.ScalismoUI
import scala.util.Random
/**
* Created by luetma00 on 07.06.16.
*/
object BasicFitting {
def main(args : Array[String]) : Unit = {
scalismo.initialize()
val model = StatismoIO.readStatismoMeshModel(new File("data/femur-ssm.h5")).get
val target = MeshIO.readMesh(new File("data/femurmesh.stl")).get
val ui = ScalismoUI()
val modelGroup = ui.createGroup("model")
val ssmView = ui.show(modelGroup, model, "femurmodel")
val shapeTransformationView = ssmView.shapeModelTransformationView.shapeTransformationView
//
val targetGroup = ui.createGroup("target")
val targetView = ui.show(targetGroup, target, "target-mesh")
targetView.color = Color.RED
def posteriorProbability(model: StatisticalMeshModel,
coefficients: DenseVector[Double],
targetMesh: TriangleMesh[_3D]): Double = ???
def propose(model: StatisticalMeshModel): DenseVector[Double] = ???
var bestProbability = Double.NegativeInfinity
for (i <- 0 until 200) {
println(s"in iteration $i")
val guess = propose(model)
val probabilityOfGuess = posteriorProbability(model, guess, target)
shapeTransformationView.coefficients = guess
println(s"probability of guess = ${probabilityOfGuess}")
bestProbability = probabilityOfGuess
}
}
}
BasicFitting
scala>

Example: 2D Gaussian

- Target: $P(\boldsymbol{x}) = \frac{1}{2\pi\sqrt{|\Sigma|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu})}$
- Proposal: $Q(\boldsymbol{x}'|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}'\,|\,\boldsymbol{x}, \sigma^2 I_2)$

Random walk

Target

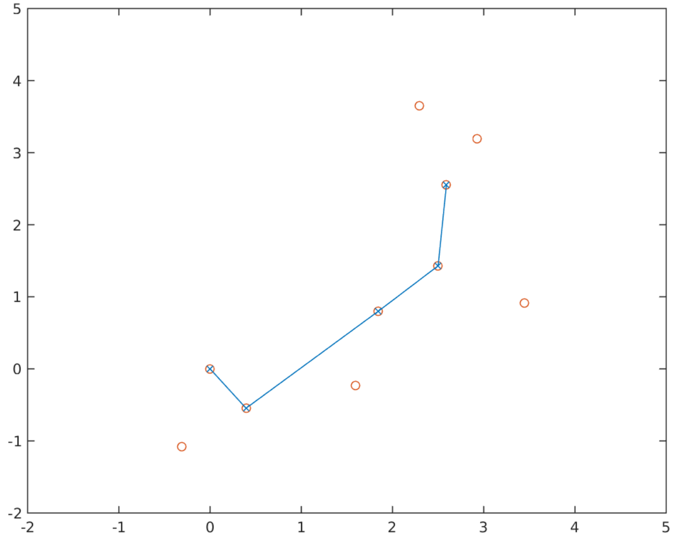
$\mu = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix}$

$\Sigma = \begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix}$

Sampled Estimate

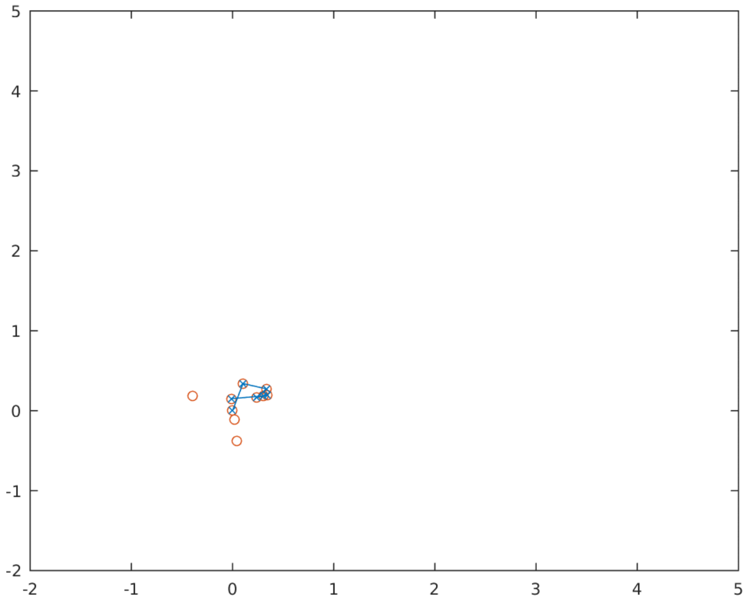
$\hat{\mu} = \begin{bmatrix} 1.56 \\ 1.68 \end{bmatrix}$

$\hat{\Sigma} = \begin{bmatrix} 1.09 & 0.63 \\ 0.63 & 1.07 \end{bmatrix}$

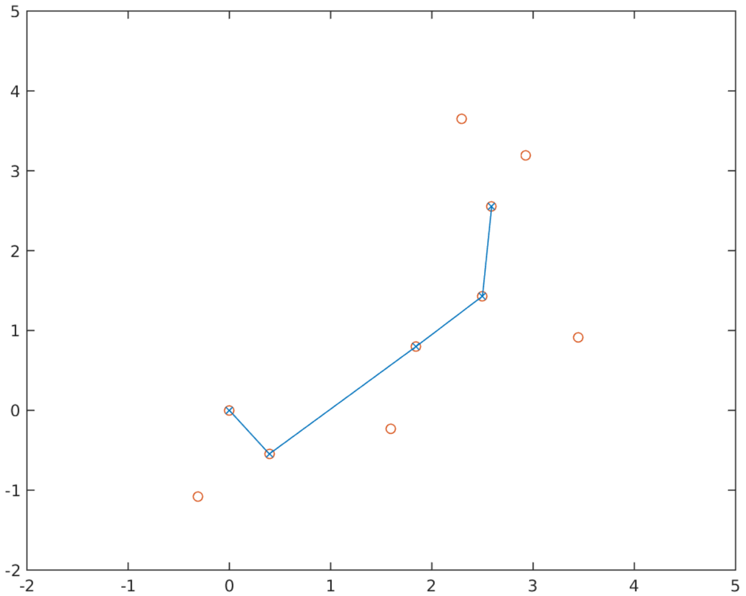


2D Gaussian: Different Proposals

$\sigma = 0.2$



$\sigma = 1.0$



The Metropolis-Hastings Algorithm

- Initialize with sample $\boldsymbol{x}$
- Generate next sample, with current sample $\boldsymbol{x}$
  1. Draw a sample $\boldsymbol{x}'$ from $Q(\boldsymbol{x}'|\boldsymbol{x})$ ("proposal")
  2. With *probability* $\alpha = \min\left\{\frac{P(x')}{P(x)}\frac{Q(x|x')}{Q(x'|x)}, 1\right\}$ accept $\boldsymbol{x}'$ as new state $\boldsymbol{x}$
  3. Emit current state $\boldsymbol{x}$ as sample

- Generalization of Metropolis algorithm to asymmetric Proposal distribution

$$Q(\boldsymbol{x}'|\boldsymbol{x}) \neq Q(\boldsymbol{x}|\boldsymbol{x}')$$

$$Q(\boldsymbol{x}'|\boldsymbol{x}) > 0 \Leftrightarrow Q(\boldsymbol{x}|\boldsymbol{x}') > 0$$

Properties

- **Approximation:** Samples $x_1, x_2, \ldots$ approximate $P(x)$

  Unbiased but correlated (not *i.i.d.*)

- **Normalization:** $P(x)$ does not need to be normalized

  Algorithm only considers ratios $P(x')/P(x)$

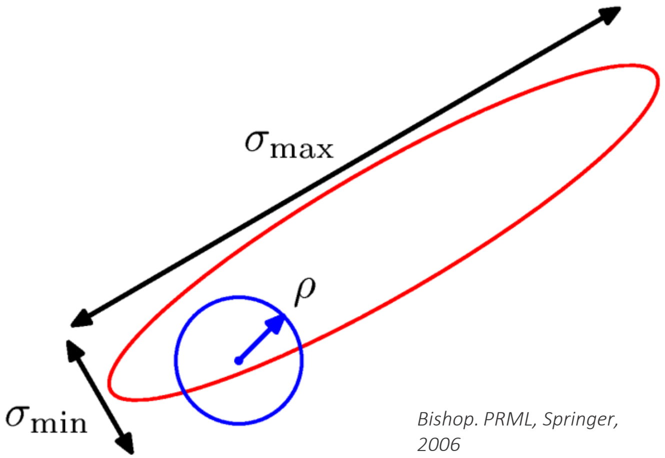
- **Dependent Proposals:** $Q(x'|x)$ depends on current sample x

  Algorithm adapts to target with simple 1-step memory

Metropolis - Hastings: Limitations

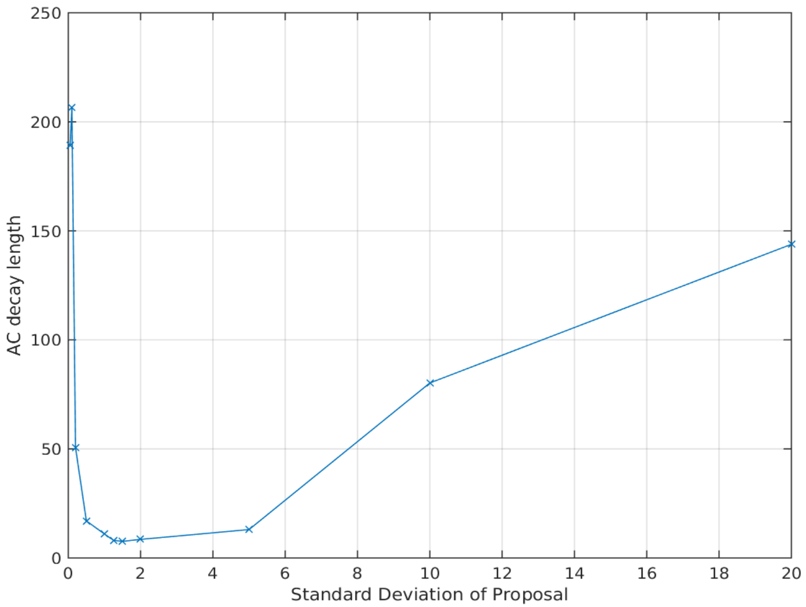
- Highly correlated targets

  Proposal should match target to avoid too many rejections



Bishop. PRML, Springer, 2006

- Serial correlation
  - Results from rejection and too small stepping
  - Subsampling



Propose-and-Verify Algorithm

- Metropolis algorithm formalizes: *propose-and-verify*
- *Steps are completely independent.*

Propose

Draw a sample x' from $Q(x'|x)$

Verify

With *probability* $\alpha = \min\left\{\frac{P(x')}{P(x)}\frac{Q(x|x')}{Q(x'|x)}, 1\right\}$ accept $\boldsymbol{x}'$ as new sample

MH as Propose and Verify

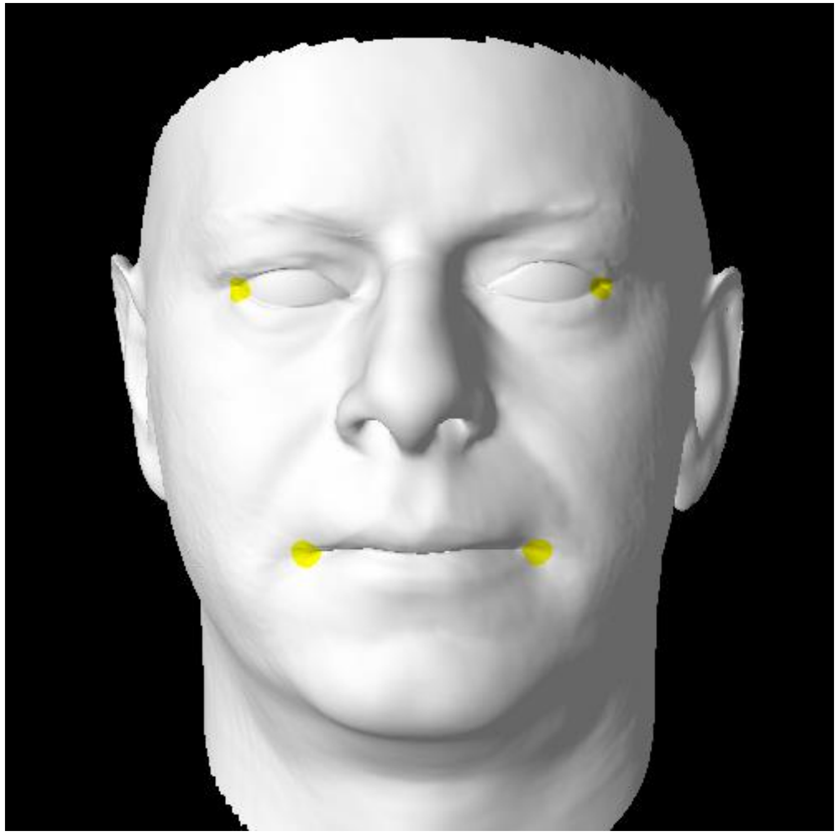
- Decouples the steps of finding the solution from validating a solution
- Natural to integrate uncertain proposals Q
  (e.g. automatically detected landmarks, ...)
- Possibility to include "local optimization" (e.g. a ICP or ASM updates, gradient step, …) as proposal

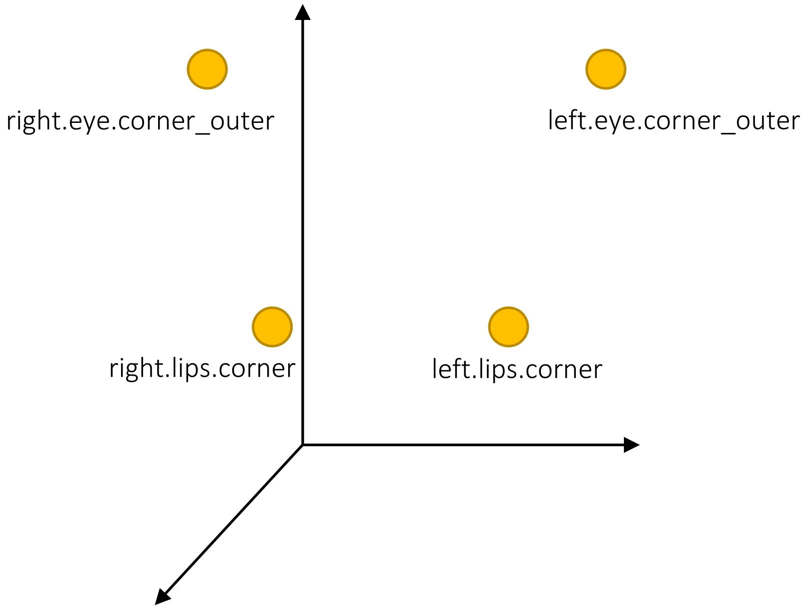
Anything more "informed" than random walk should improve convergence.

Fitting 3D Landmarks

3D Alignment with Shape and Pose

3D Fitting Example





3D Fitting Setup

Goal: *Find posterior distribution for arbitrary pose and shape*

Shape transformation

$$\varphi_S[\alpha] = \mu(x) + \sum_{i=1}^{r} \alpha_i \sqrt{\lambda_i} \Phi_i(x)$$

Rigid transformation

- 3 angles (pitch, yaw, roll) φ, ψ, ϑ
- Translation $t = (t_x, t_y, t_z)$

$$\varphi_R[\varphi, \psi, \vartheta, t] = R_\vartheta R_\psi R_\varphi(\boldsymbol{x}) + t$$

Full transformation

$$\varphi[\theta](x) = (\varphi_R \circ \varphi_S)[\theta](x)$$

Observations

- Observed positions $l_T^1, \dots, l_T^n$
- Correspondence: $l_R^1, \dots, l_R^n$

Parameters

$$\theta = (\alpha, \varphi, \psi, \vartheta, t)$$

Posterior distribution:

$$P(\theta \middle| l_T^1, \dots, l_T^n) \propto p(l_T^1, \dots, l_T^R | \theta) P(\theta)$$

Proposals

- Gaussian random walk proposals

$$"Q(\theta'|\theta) = N(\theta'|\theta, \Sigma_\theta)"$$

- Update different parameter types block-wise
    - Shape $N(\boldsymbol{\alpha}'|\boldsymbol{\alpha}, \sigma_S^2 I_{m\times m})$
    - Rotation $N(\varphi'|\varphi, \sigma_\varphi^2),\ N(\psi'|\psi, \sigma_\psi^2), N(\vartheta'|\vartheta, \sigma_\vartheta^2)$
    - Translation $N(\boldsymbol{t}'|\boldsymbol{t}, \sigma_t^2 I_{3\times 3})$

- Large mixture distributions as proposals
    - Choose proposal Q_i with probability c_i

$$Q(\theta'|\theta) = \sum c_i Q_i(\theta'|\theta)$$

3DMM Landmarks Likelihood

Simple models: **Independent Gaussians**

Observation of L landmark locations l_T^i in image

- Single *landmark position* model:

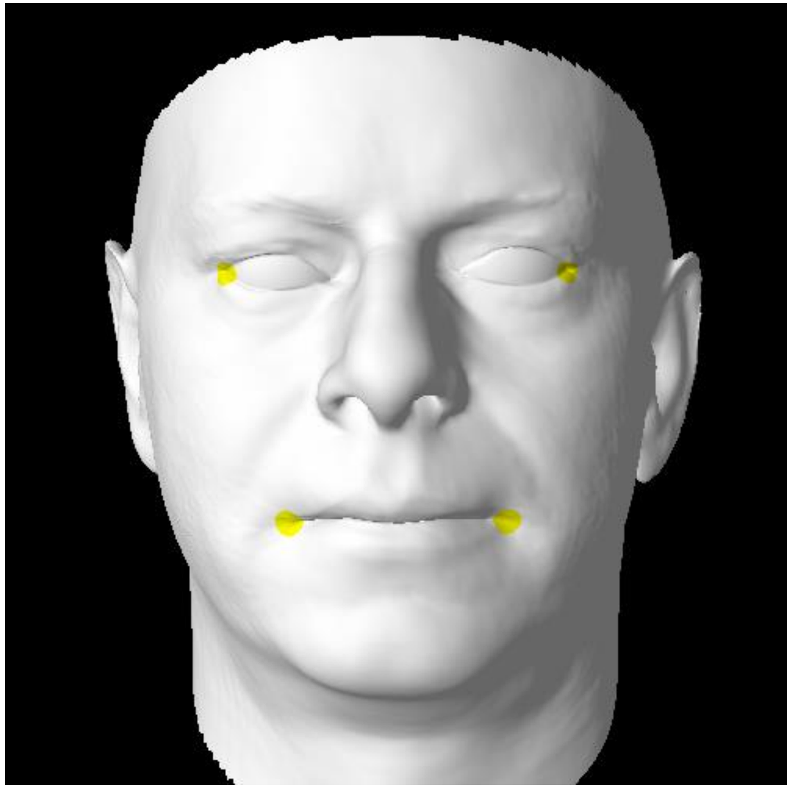
$$p(l_T|\theta, l_R) = N(\varphi[\theta](l_R), I_{3\times3}\sigma^2)$$

- *Independent* model (conditional independence):

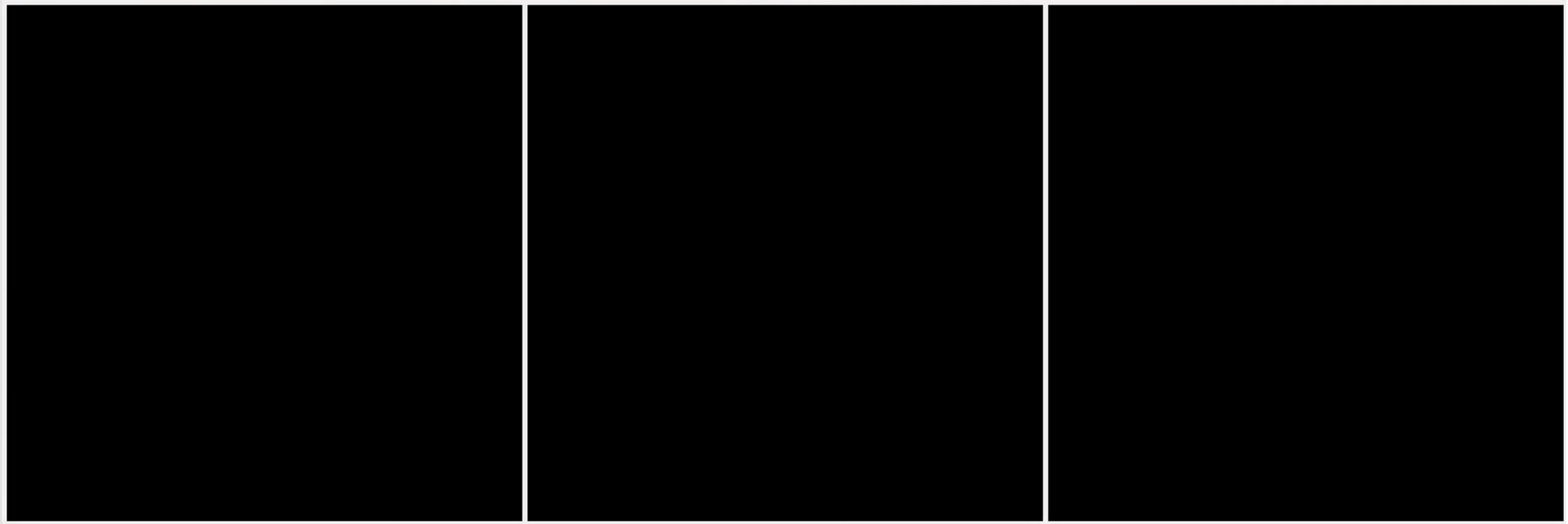
$$p(l_T^1, \dots, l_T^n|\theta) = \prod_{i=1}^{L} p_i\left(l_T^i|\theta\right)$$

3D Fit to landmarks

- Influence of landmarks uncertainty on final posterior?
    - $\sigma_{\text{LM}} = 1\text{mm}$
    - $\sigma_{\text{LM}} = 4\text{mm}$
    - $\sigma_{\text{LM}} = 10\text{mm}$
- Only 4 landmark observations:
    - Expect only weak shape impact
    - Should still constrain pose
- Uncertain landmarks should be looser



Posterior: Pose & Shape, 4mm



$\hat{\mu}_{\mathrm{yaw}} = 0.511$ $\hat{\mu}_{\mathrm{t_x}} = -1\ \mathrm{mm}$ $\hat{\mu}_{\alpha_1} = 0.4$

$\hat{\sigma}_{\mathrm{yaw}} = 0.073\ (4°)$ $\hat{\sigma}_{\mathrm{t_x}} = 4\ \mathrm{mm}$ $\hat{\sigma}_{\alpha_1} = 0.6$

(Estimation from samples)

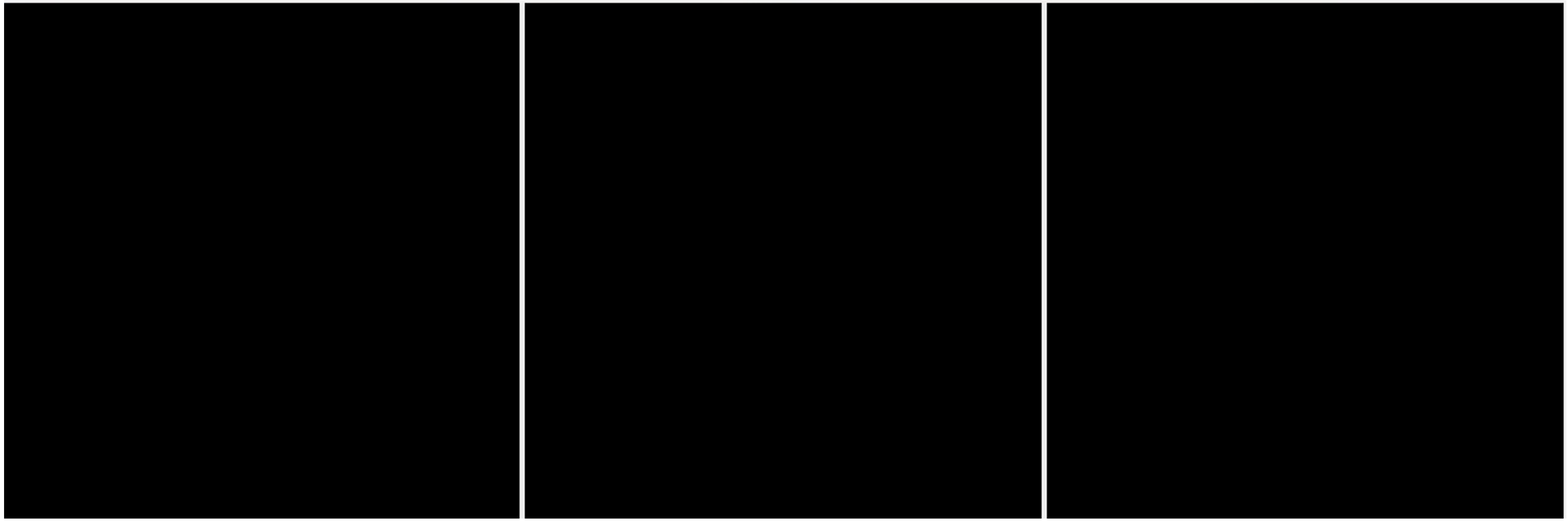
Posterior: Pose & Shape, 1mm

$\hat{\mu}_{\text{yaw}} = 0.50$
$\hat{\sigma}_{\text{yaw}} = 0.041\ (2.4°)$

$\hat{\mu}_{\text{t}_\text{x}} = -2\text{ mm}$
$\hat{\sigma}_{\text{t}_\text{x}} = 0.8\text{ mm}$

$\hat{\mu}_{\alpha_1} = 1.5$
$\hat{\sigma}_{\alpha_1} = 0.35$

Posterior: Pose & Shape, 10mm



$\hat{\mu}_{\mathrm{yaw}} = 0.49$
$\hat{\sigma}_{\mathrm{yaw}} = 0.11\ (7°)$

$\hat{\mu}_{\mathrm{t_x}} = -5\ \mathrm{mm}$
$\hat{\sigma}_{\mathrm{t_x}} = 10\ \mathrm{mm}$

$\hat{\mu}_{\alpha_1} = 0$
$\hat{\sigma}_{\alpha_1} = 0.6$

Summary: MCMC for 3D Fitting

- Probabilistic inference for fitting probabilistic models
  - Bayesian inference: posterior distribution
- Probabilistic inference is often intractable
  - Use *approximate* inference methods
- MCMC methods provide a powerful sampling framework
  - Metropolis-Hastings algorithm
    - Propose update step
    - Verify and accept with probability
- Samples converge to true distribution: More about this later!