Sampling II

{Sidenote: if you scroll through the past lectures, you will see I often created fake data for the lecture. [Fake stars, fake telescope noise, fake random fields.] It is these sampling techniques which do the maths behind those pictures.}

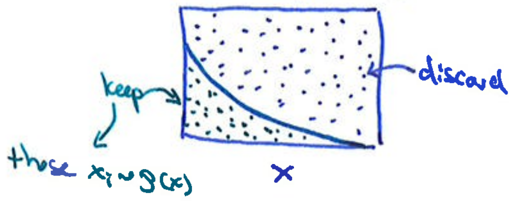
- we had the basics of sampling:

1) $m(\vec{\theta}) \propto \mathcal{P}(\vec{\theta})$

2) if $\underset{\text{samples}}{x_i} \sim \mathcal{P}(x|M)$ then $\int dx\, \mathcal{P}(x|M)\, g(x) \approx \frac{1}{N} \sum_{n=1}^{N} g(\underset{\text{samples}}{x_n})$

3) Markov chain: x_i depends only on x_{i-1}, not x_{i-n} with $n>1$.

4) Sampling via rejection sampling or by transformation of probabilities



$$\mathcal{P}(x)\,dx = \mathcal{P}(y)\,dy$$

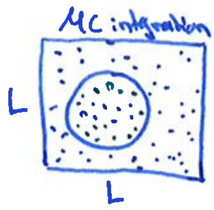
Mostly, these algorithms occur simultaneously / in conjunction / and protocoll other interesting variables along the way.

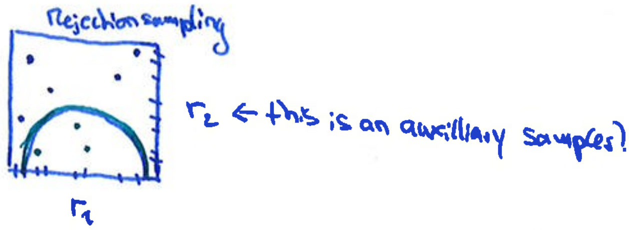
Example: the chains you analyzed in the tutorials protocoled weights, χ^2 and log-likelihoods as auxilliary variables.

Q: what is the difference between Monte Carlo Integration and rejection sampling?

A: 1) They solve different problems:
- MC-Integration happens to use random numbers, but would like the resulting integral to be a „classical" (noise free) value. $\rightsquigarrow N \to \infty$
- Rejection sampling ~~also~~ generates random numbers which follow a distribution.

2) Also the algorithms differ slightly:





$\rightsquigarrow$ 2D Integral

$\rightsquigarrow$ computes fraction inside/outside

$A_{square} = L^2$

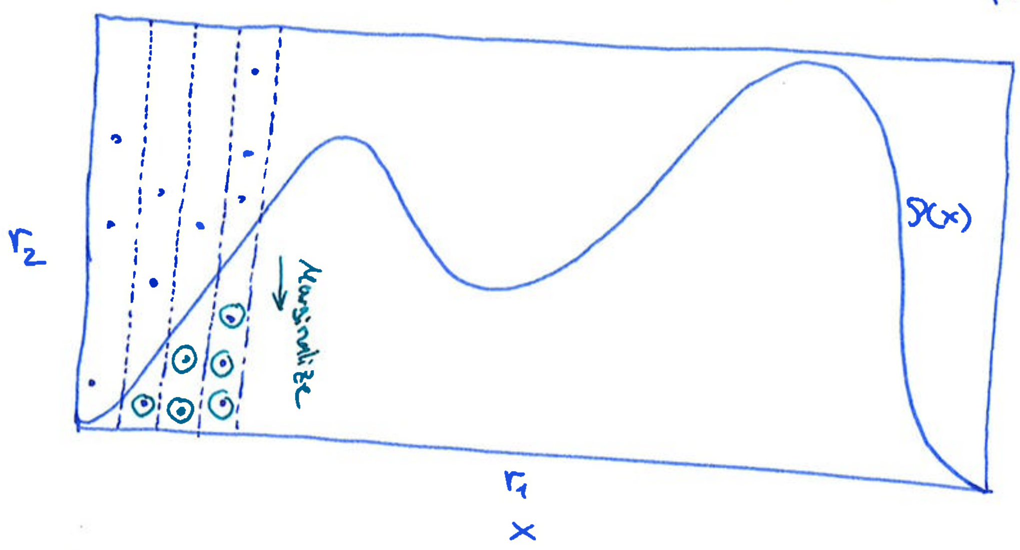
$A_{circle} = ?$

$\rightsquigarrow$ 2D sampling, but 1D distribution

$\rightsquigarrow$ Aim: samples that follow the turquoise (semi-circle) distribution

· Rejection sampling, zoomed in: why does it work?

Had: draw x, compute $P(x)$. Draw $q \sim \mathrm{Unif}[0,1]$. If $P(x) > q$, keep x.
[also occurs in the accept/reject step of Metropolis algorithm for MCMC]



$$P(r_1, r_2) = P(r_1)\,P(r_2)$$

$$\rightsquigarrow P(r_1) = \int_0^1 P(r_1, r_2)\, dr_2 = P(r_1)\cdot 1 \qquad \text{because } \int P(r_2)\, dr_2 = 1$$

But now we keep only those r_2s below the curve $P(x)$

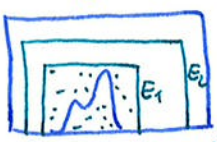
$$\int_0^{P(x)} P(r_1, r_2)\, dr_2 = \int_0^{P(x)} P(r_1)\, P(r_2)\, dr_2 = P(x)$$

· Gibbs-Sampling: a sampler without tuneable parameters (good!)

(also a sampler which let's you get around evaluating $P(x)$ [$\propto \exp(\dots)$], which can be under- and over-flow unstable)

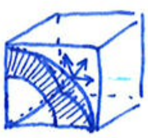
· Tunable parameters:

1) Rejection sampling:
   - maximum of $P(x)$ needs to be estimated
   - all samples need to be discarded if $P > P_{max}^{thought}$ is found [i.e. maximum was not correct)
   - more general: sampling envelope needs to be tuned for efficiency



$\rightsquigarrow$ envelope E_1 more efficient than E_2

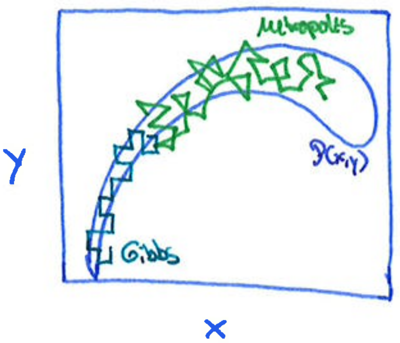
2) Metropolis algorithm:
   $\vec{\Delta}$, the proposal step width and direction



$\rightsquigarrow$ for „wrong" $\vec{\Delta}$, the acceptance ratio will drastically decrease.

~> the Gibbs sampler does not have free parameters which affect its performance.

+: for many problems, Gibbs sampling is naturally highly effective (few correlations)



— Metropolis MCMC Algorithm

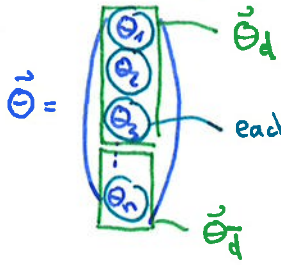
— Gibbs Sampler

~> it walks „along the axes"

· Preconditions for Gibbs-Sampling: If $P(x,y,z)$ is to be sampled, then all conditionals need to be known.

· Let $\vec{\Theta}$ be the collection of all random variables in the game.

~> Ways to split up $\vec{\Theta}$:



+ Need all $P(\vec{\Theta}_d \mid \vec{\Theta}_{\bar{d}})$

Gibbs sampling in Pseudo code:

0) initialize $\vec{\Theta}$

1) For $i=0,\dots,N_{samples}$:

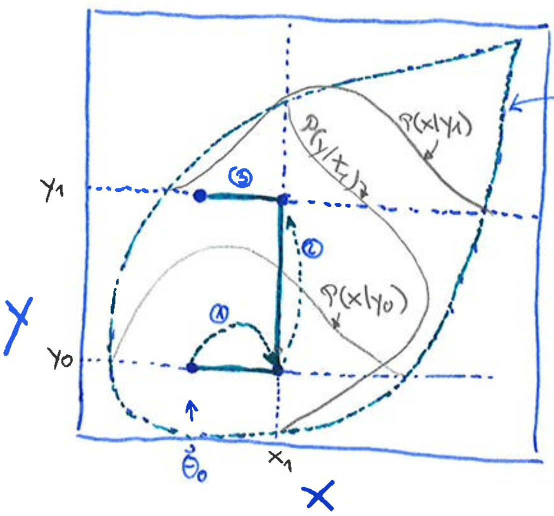
a) $\forall d$ draw $\vec{\Theta}_d \sim P(\vec{\Theta}_d \mid \vec{\Theta}_{\bar{d}})$

b) replace elements of $\vec{\Theta}$ with $\vec{\Theta}_d$

c) [randomize in which order components d are drawn]

for performance reasons

Example:



yet unknown joint distribution $P(x,y)$

~> there is no $\frac{P(\vec{\Theta}_*)}{P(\vec{\Theta}_i)} \leq \alpha$, hence one does not need to evaluate the P function, as long as one is able to generate samples from it

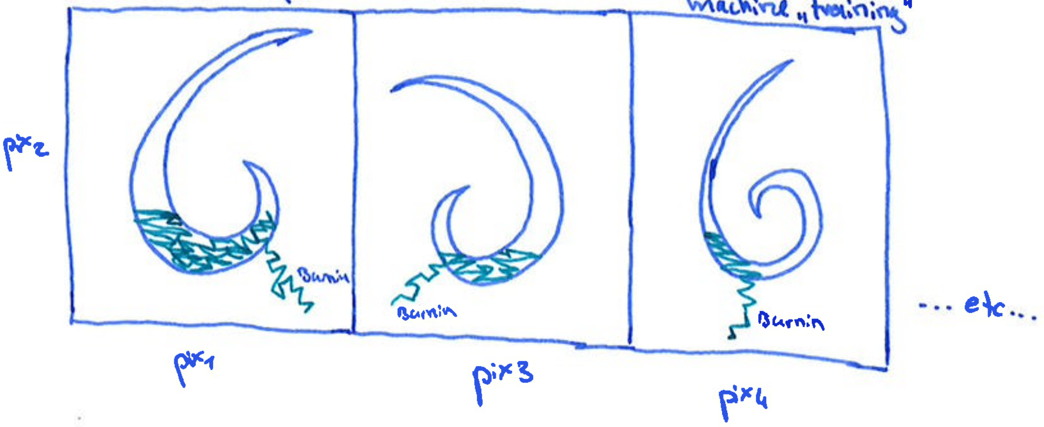
(often true, e.g. for χ^2: $x \sim \chi^2_p$: $x = \sum_{i=1}^{p} x_i^2$, $x_i \sim G(0,\mathbb{1})$)

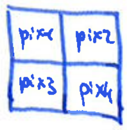
· The No 1 problem with sampling: when samples are not representative of the distribution

⇝ it is the same problem as faced by machine „learning" algorithms, when confronted with unrepresentative samples.

actually it's machine „training"

Generate a field



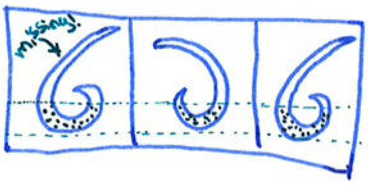


2x2 image

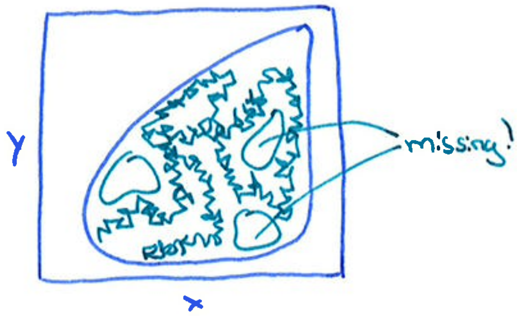
1) Burnin not representative

2) Sampler get's stuck: it doesn't suceed to get around the curve.

Result: it generates samples which capture only part of the variability.

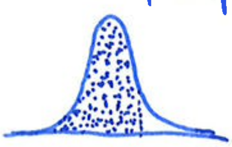


Another problem are strong correlations in chains.



⇝ If a artificial neural network were now trained on those 2x2 images, then it would not „learn" (= „fit to") the true distribution. Because unrepresentative samples cause biases.

Extremely simple case:

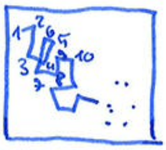


⇒ obviously one Gaussian tail is missing

· The #1 user-error for using samples which are representative of the true distribution:

⇝ using neighbouring samples ← don't!

Example: MCMC:



2 depends on 1
3 depends on 2
4 depends on 3
⋮
10 does not depend on 1
100 does not depend on 2

} user selects samples 1,2,3.
⇓
user selection induces errors, although sampler ran fine.

French: together! All samples count!

⇝ Samples form an ensemble. If you use samples from an MCMC chain for post processing, Pick randomly from the chain!

· The #2 user error of MCMC sampling: concatenating unconverged chains

⇝ Converged chain: $n_1 \propto \rho$

⇝ 2nd converged chain: $n_2 \propto \rho$

} $n_2 \propto n_1$ since both converged
⇝ can concatenate (= „paste both into same .txt file")

⇝ 1st unconverged chain: $n_1 \not\propto \rho$

⇝ 2nd unconverged chain: $n_2 \not\propto \rho$

} concatenation does not improve the quality of the chain! chain1 + chain2 still not $\propto \rho$!

Example:

⇝ chain1 + chain2 still misses middle part of distribution

Mini-crash course: Machine „Learning"

Training

~~Distribution~~ => Samples => „Learning from Examples"

then called „training data"

traditional inference

since we gave up on the distribution [in general] replace it by „a loss function"

if $D(x)$ is the sampling distribution of data x, and I want to learn about parameters θ, then

$D(x|\theta)$ is my likelyhood.

$\partial_\theta D(x|\theta) \overset{!}{=} 0$ then determines my best fit

Training data + lossfunction

$\boxed{\text{Many free tunable parameters} \uparrow \text{„learnable"}}$ ← „the Machine" e.g. neural net, SVM, etc

⇓

output: values of the tunable parameters, for which the loss function is small.

⇓

Save to disk.

Simplemost example:

$$A \vec{x}_i \overset{!}{=} \vec{b}_i \quad \forall i$$

the „machine"

data

the physics which generated it

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{12} & A_{22} & A_{23} & A_{24} \\ A_{13} & A_{23} & A_{33} & A_{34} \\ A_{14} & A_{24} & A_{34} & A_{44} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \overset{!}{=} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

⊕ Loss function: e.g. $|\vec{b}_M - \vec{b}_{true}|$

the „learnable" (= „fittable") parameters

many trainings data

many provided wanted outcomes

Aim: find A such that it maps $\vec{x}_i$ correctly to $\vec{b}_i$, by fitting the A_{ij} elements.

⇝ obviously this is mathematically solvable (linear algebra), hence „learnable" from unbiased examples.

After N → many training runs with N examples, feed in one more data vector, this time the true one. Since $\frac{N+1}{N}$ small (because $N \gg 1$), loss will again be small (if all went well).

⇝ it „learned sth".