STAN workshop/tutorial

Sergey Koposov

Outline

- Reminder Bayesian Statistics
- Markov Chain Monte-Carlo, existing methods
- Hamiltonian Monte-Carlo (No-U-Turn-Sampling)
- Stan code
- Simple models
- Hierarchical models.
- Astrophysically motivated models
- Variational Inference

Bayesian statistics

- D – data
- φ – parameters
- $P(D|\varphi)$ – Probability of the data given the parameters (also known as likelihood function)
- $P(\varphi|D)$ – Probability distribution over-parameters after obtaining data (posterior distribution)
- $\pi(\varphi)$ – Prior probability, probability distribution over parameters before obtaining data (either subjective or incorporating previous observations)

Bayes Theorem

- Bayes theorem connects the PDF over-parameters before obtaining data to the PDF when data was obtained

$$P(\phi|D) = \frac{P(D|\phi)\pi(\phi)}{P(D)}$$

$$P(\phi|D) \propto P(D|\phi)\pi(\phi)$$

Generative model

- Likelihood function P(D|φ)
- PDF over data.
- You can generate replicated datasets

Thinking about a generative model for your data.

- Example dataset: Positions of stars, magnitudes/colors of stars, radial velocities

  $\alpha_i, \delta_i, mag_i, V_{rad,i}$

- One way to specify a generative model:

  $P(\alpha, \delta, mag, V_{rad} | parameters)$

- Or

$$\begin{aligned}
\alpha &\sim Uniform(\alpha_0, \alpha_1) \\
\delta &\sim Uniform(\delta_0, \delta_1) \\
mag &\sim Normal(m_0, \sigma) \\
V_{rad} &\sim Normal(F(mag), \sigma_v)
\end{aligned}$$

Thinking about a generative model for your data.

- Example dataset: Positions of stars, magnitudes/colors of stars, radial velocities

$$\alpha_i, \delta_i, mag_i, V_{rad,i}$$

- One way to specify a generative model:

$$P(\alpha, \delta, mag, V_{rad} | parameters)$$

- Or

$$\begin{aligned} \alpha &\sim Uniform(\alpha_0, \alpha_1) \\ \delta &\sim Uniform(\delta_0, \delta_1) \\ mag &\sim Normal(m_0, \sigma) \\ V_{rad} &\sim Normal(F(mag), \sigma_v) \end{aligned}$$

Connections to standard least squares

- $\{x_i, y_i\}$ where y_i is measured with error e_i
- $y = F(x, \phi)$
- $y_i \sim \mathcal{N}(F(x_i, \phi), e_i)$
- $P(y|\phi) \propto \exp\left[-\frac{1}{2}\sum\left(\frac{y_i - F(x_i, \phi)}{e_i}\right)\right]$
-
- $\log \mathcal{L} = -\frac{1}{2}\sum\left(\frac{y_i - F(x_i, \phi)}{e_i}\right)$
- Maximising this likelihood function w.r.t. φ is equivalent to doing least-squares.

Finding posterior distribution

- $P(\phi|D) \propto P(D|\phi)\pi(\phi)$
- Closed form analytical expressions – unlikely except in simplest problems
- We also often care about marginalisations

$$P(\phi_1|D) = \int P(\phi_{1:n}|D) d\phi_2 ... d\phi_n$$

- And expectations $E[F] = \int F(\phi) P(\phi_{1:n}|D) d\phi_{1:n}$
- Solution: represent the distribution by samples from it.

$$P(\phi|D) \to \{\phi^1, \phi^2 .... \phi^N\}$$

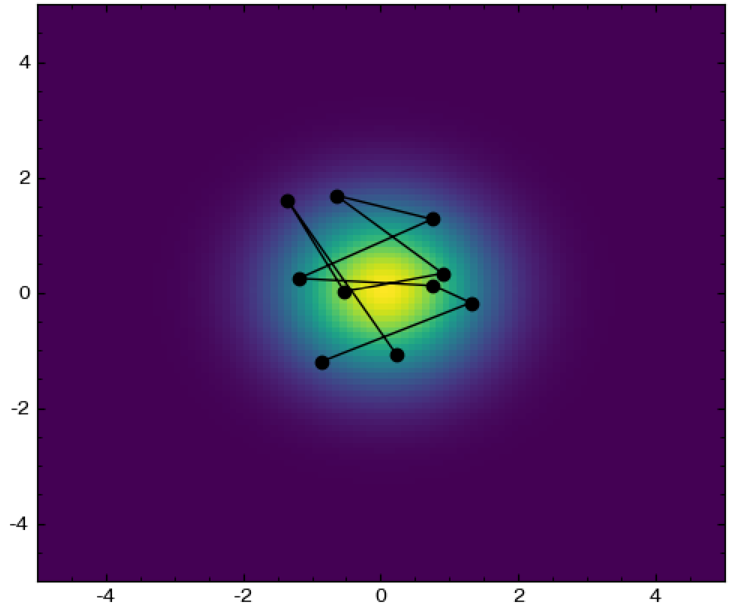
Markov-Chain Monte-Carlo

- Target distribution $\pi(x)$
- Markov Chain $x_{t-1} \rightarrow x_t$
- If Markov Chain satisfies detailed balance:

$$\pi(x)P(x \rightarrow y) = \pi(y)P(y \rightarrow x)$$

samples from the chain will converge to $\pi(x)$

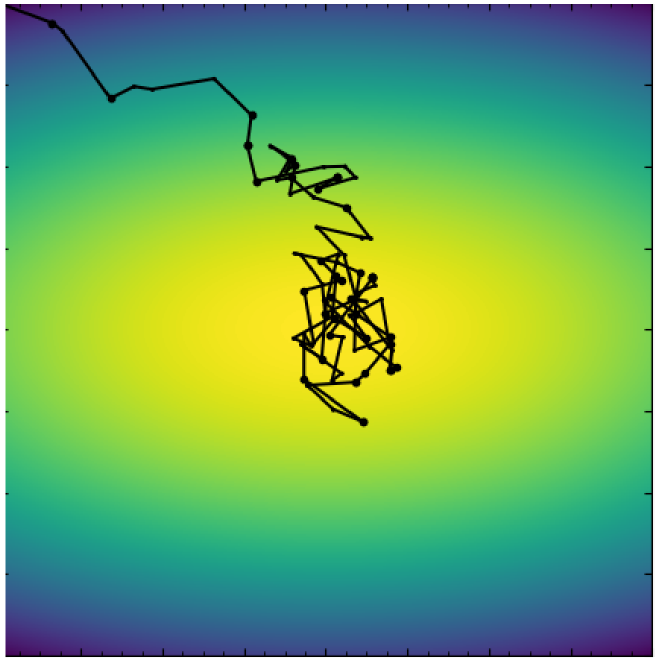
(no matter what is $P(x \rightarrow y)$)



Existing sampling methods

- Gibbs sampling
- Metropolis-Hastings (Hastings, Ulam)
- Ensemble sampling (Goodman, Weare)
- Nested sampling (Skilling)
- Hamiltonian Monte-Carlo (Neal)

Metropolis-Hastings algorithm



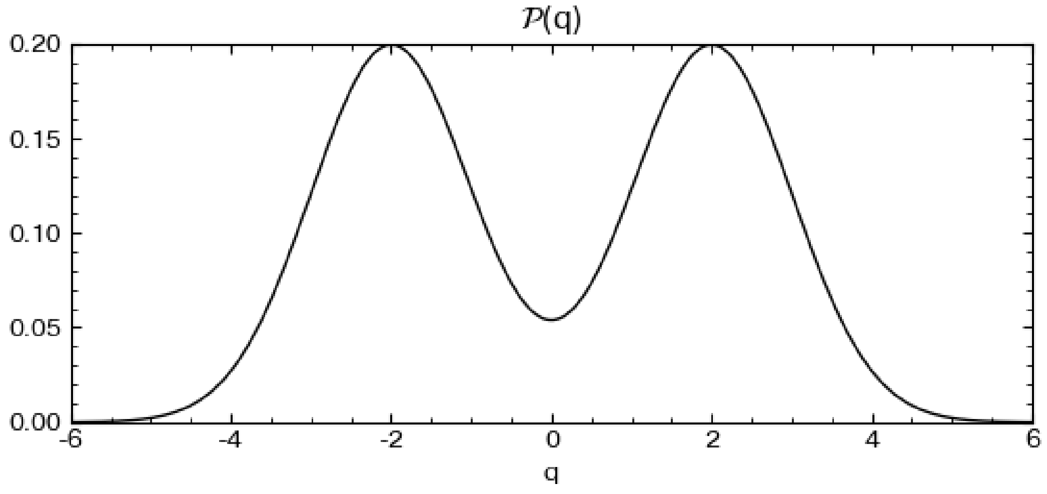
- At each step propose a new point by sampling from a distribution (i.e. Normal)
- $x_{t+1,proposed} \sim \mathcal{N}(x_t, \Sigma)$
- Accept a new point with probability: $min(1, \pi(x_{t+1})/\pi(x_t))$
- Guaranteed to converge to target distribution
- Leads to random walk behaviour: Inefficient in large number of dimensions

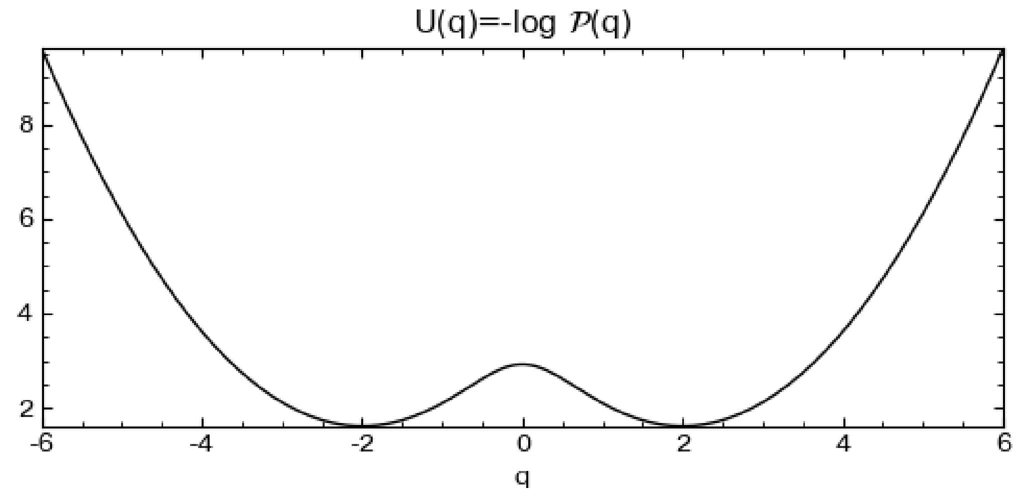
Hamiltonian Monte-Carlo

- Neal (arxiv: 1206.1901), see also introduction arxiv: 1701.02434
- Monte-Carlo algorithm using gradients to avoid random-walk behaviour
- Reason for popularity – Autodiff and symbolic differentiation

Posterior as potential energy

- Posterior $\mathcal{P}(q)$
- Construct the potential energy function
  $U(q) = -\ln \mathcal{P}(q)$
- Consider motion in this potential



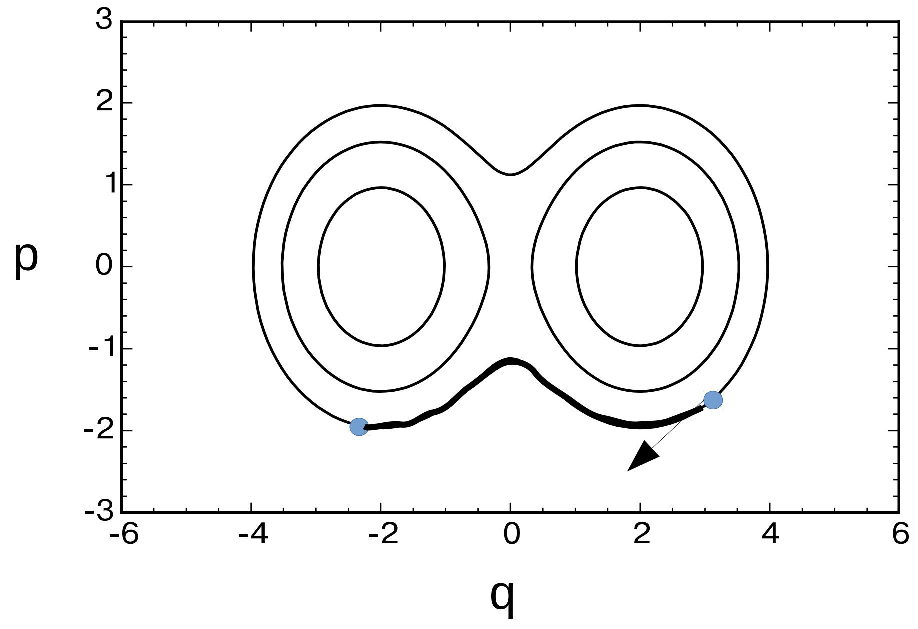


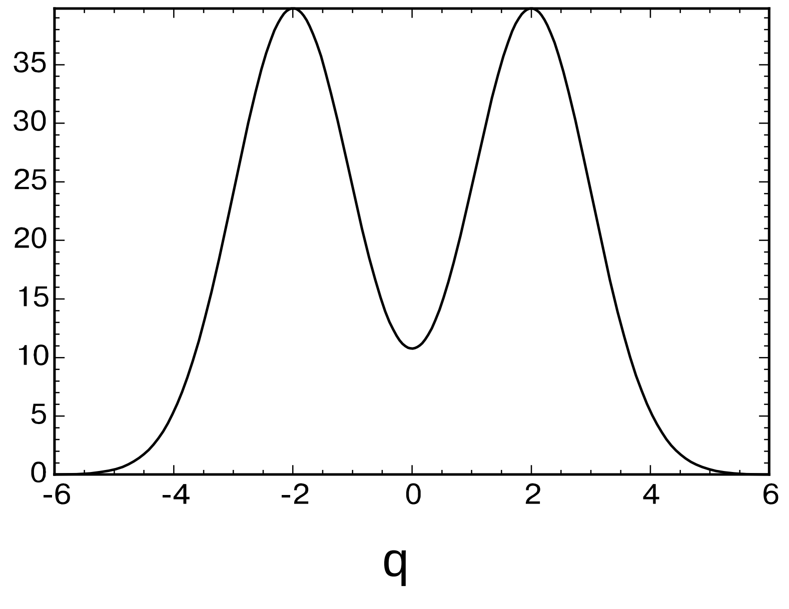
Hamiltonian Monte-Carlo

- We can use gradients of the posterior for faster exploration !
- Posterior: $\mathcal{P}(q)$
- Augment it by a “dummy” (momentum/velocity) variable with Normal distribution
- $\mathcal{P}(q,p) \propto \mathcal{P}(q)\exp(-p^2/2) = \exp(-p^2/2 + \ln\mathcal{P}(q))$
- Boltzmann distribution with Hamiltonian (sum of kinetic and potential energy)
  $H(p,q) = p^2/2 - \ln\mathcal{P}(q)$

Hamiltonian Monte-Carlo

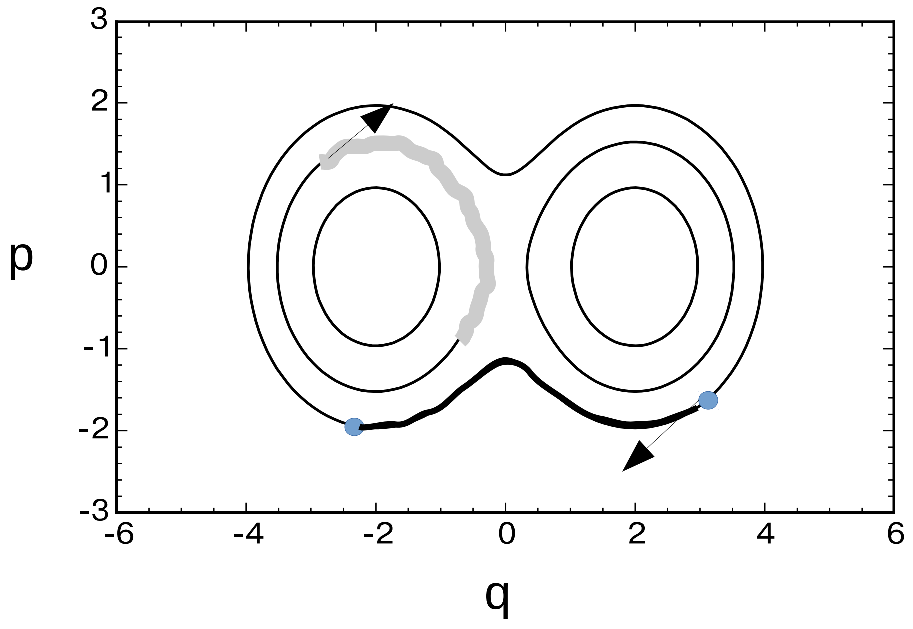
- For a Hamiltonian we know the equations of Motion!
- Equation of motion preserves energy
- We move along the level of constant P(p,q)
- $$\frac{dp}{dt} = -\frac{\partial H(p,q)}{\partial q} = \frac{\partial \ln P(q)}{\partial q}$$
  $$\frac{dq}{dt} = \frac{\partial H(p,q)}{\partial p} = p$$

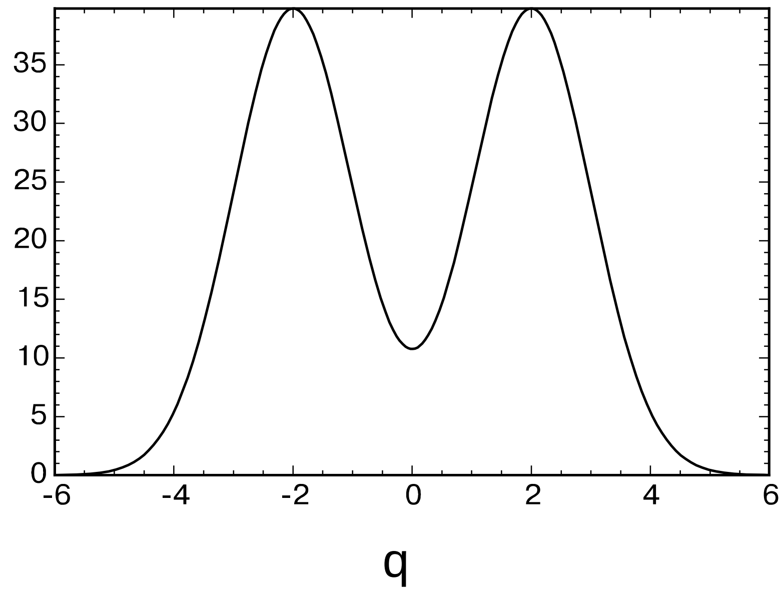




Hamiltonian Monte-Carlo

- At every step randomly sample momentum/velocity
- p ~ N(0,1)
- Move from current point (q, p) to (q_{new},p_{new}) following Hamilton's equations
- Energy is conserved!
- Randomness is avoided!





Hamiltonian Monte-Carlo

- At every step randomly sample momentum/velocity
- p ~ N(0,1)
- Move from current point (q, p) to (q_{new},p_{new},p following Hamilton's equations using leapfrog).

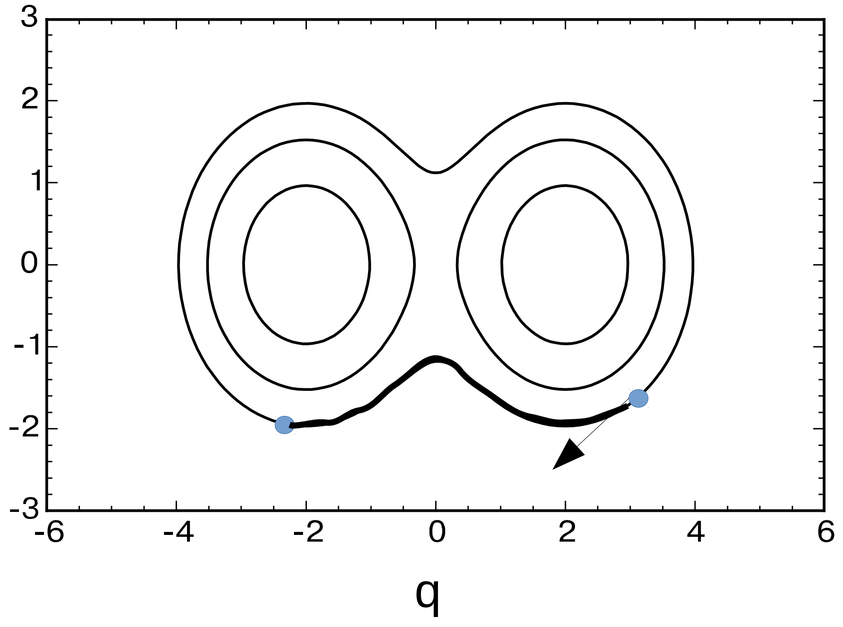
$$\mathrm{p}_i(t+\epsilon/2) = p_i(t) - \epsilon/2\frac{\partial H(p,q)}{\partial q}$$

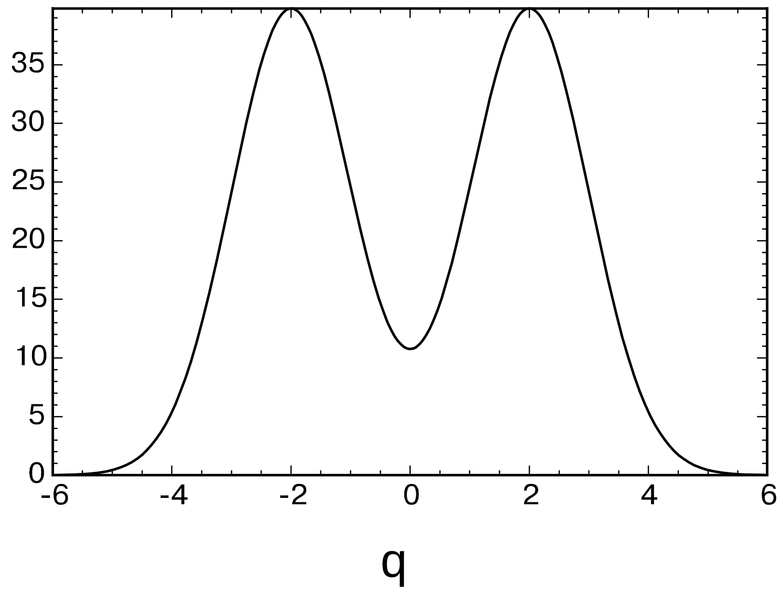
$$\mathrm{q}_i(t+\epsilon) = q_i(t) + \epsilon v_i(t+\epsilon/2)$$

$$\mathrm{p}_i(t+\epsilon) = v_i(t+\epsilon/2) - \epsilon/2\frac{\partial H(p,q)}{\partial q}$$

- Energy is approximately conserved!
- Randomness is avoided!
- Accept or reject the new point

  with probability min(1, exp(E_{new}-E_{old}))



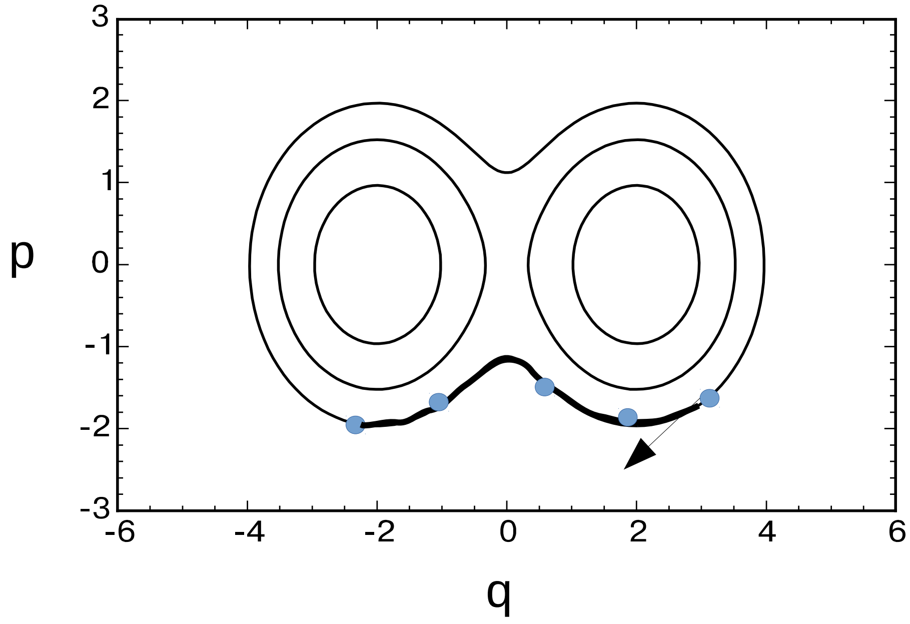


Hamiltonian Monte-Carlo, tuning

- Tuning parameters: Step size (ε)
- Number of steps along the trajectory L
- Mass matrix

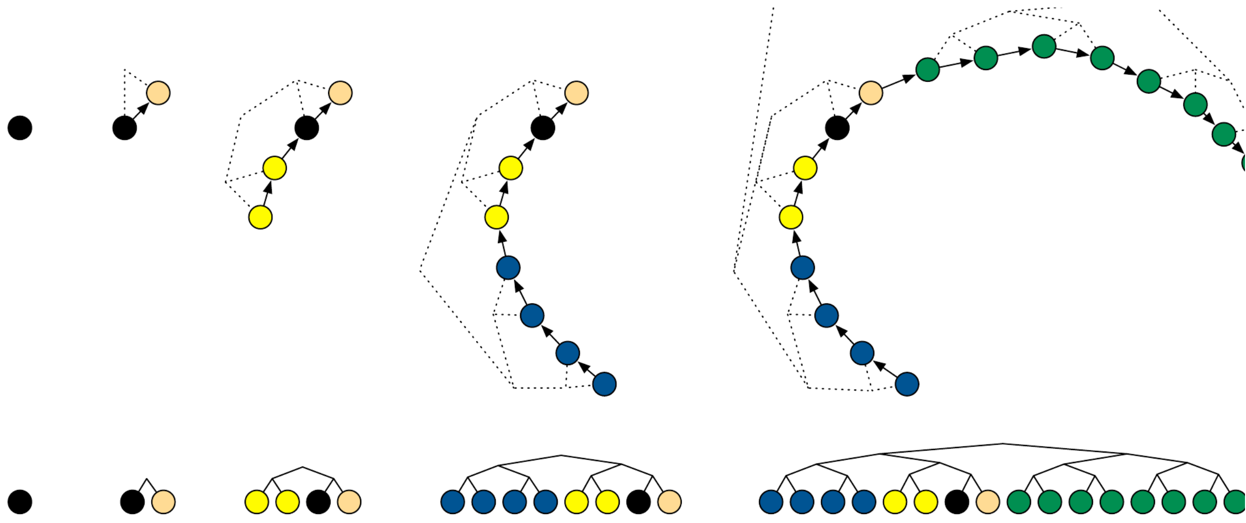
$$H(p,q) = \sum \frac{p_i^2}{2m_i} - \ln \mathcal{P}(q)$$

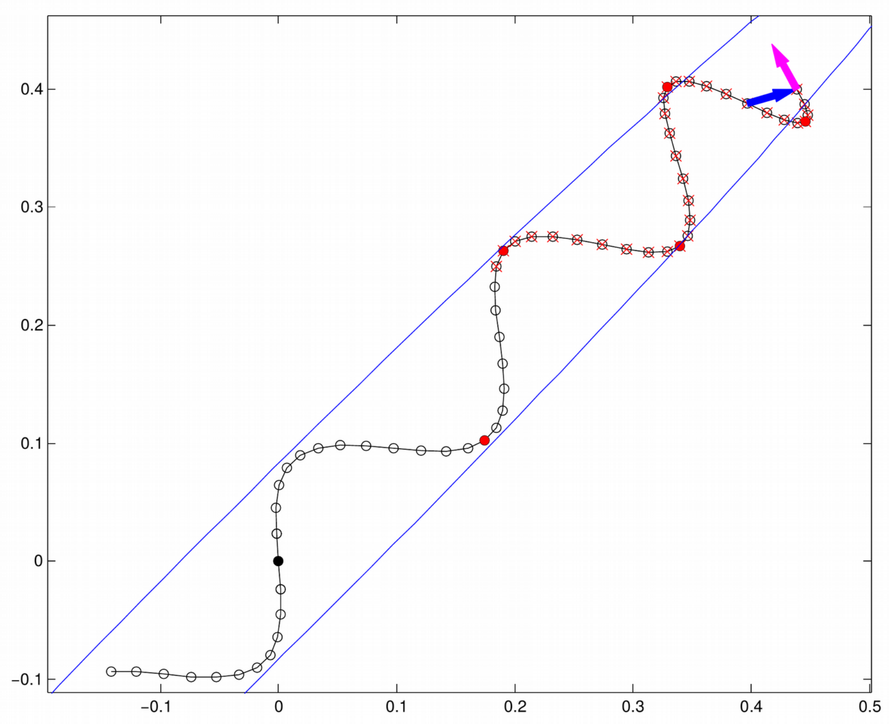
masses should reflect the scales of different parameters



No-U-Turn Sampler

- Crucial parameters of HMC are ε, L
- How to chose them (arxiv:1111.4246 Hoffman, Gelman)
- Iteratively Double the trajectory length till the U-turn. Select some point along the trajectory





Stan

- Probabilistic Programming language implementing HMC and NUTS (Inference engine)
- http://mc-stan.org
- Carpenter + 2016 (DOI 10.18637/jss.v076.i01)
- Name from Stanislaw Ulam (one of the MCMC inventors)
- Open-Source
- Well tested
- Written by professional statisticians/computer scientists

Alternatives

- HMC codes:

  Pymc3 - Hamiltonian Monte-Carlo + NUTS (python using theano as backend)

  Edward - Hamiltonian Monte-Carlo (python using tensorflow as a backend) (arxiv:1610.09787)

- Other powerful samplers that do not use the gradients
- Multinest (and other nested sampling implementations)
- Emcee (Parallel Tempering sampler of Emcee)

Prerequisites for the workshop

- If you are using Python

  $ pip install --user pystan
- If you are planning to use CmdStan download it here http://mc-stan.org/interfaces/cmdstan
- Goto for further instructions for the tutorial http://github.com/segasai/stan_workshop/README.md

Stan ingredients

- Model language – programming language to define likelihood/posterior/prior functions
- Different method how to sample or optimize or approximate likelihoods/posteriors.

Key blocks

- Order of blocks is fixed
- Functions – New function definition
- Data – input data
- Transformed data – any data transformations (done once) optional
- Parameters – The list of model parameters
- Transformed parameters (optional)
- Model – sampling statements
- Generated quantities (optional)

```
functions {
..
}
data{
…
}
transformed data {
...
}
parameters {
...
}
transformed parameters {
...
}
model {
..
}
generated quantities {
...
}
```

Information about blocks

	data	Transformed data	parameters	Transformed parameters	model	Generated quantities
Execution	Per chain	Per chain	-	Per leapfrog	Per leapfrog	Per sample
Variable declaration	Yes	Yes	Yes	Yes	Yes	Yes
Variable scope	Global	Global	Global	Global	Local	Local
Variables Saved	No	No	Yes	Yes	No	Yes
Modify posterior	No	No	No	No	Yes	No
Random variables	No	No	No	No	No	Yes

Types

- Standard c/c++ types int, real (8 bit floating point)
- Arrays int[] , real[] `Int x[30];`
- Vectors, matrices (they are ONLY floating point):

```
vector[30] x;
simplex[3] y;
unit_vector[4] z;
```

- Constrained types unit_vector, simplex, ordered, covariance_matrix
- Arrays are 1-based!
- Arrays of arrays `matrix[20,20] A[10];`

Type limits

- Lower, upper
- All the parameters are internally rescaled to the unconstrained domain (important for HMC)
-
```
int <lower=0> x;
real <upper=5> y;
real <lower=-2,upper=100> z[100];
```
-
- Both parameters and data could be constrained

Sampling statements

- Large number of Probability distributions (see manual)

```
x ~ poisson(lambda)
y ~ normal(mu, sigma)

// Equivalent to adding to logposterior

target += poisson_lpmf(x| lambda);
target += normal_lpdf(y| mu, sigma);

// target is the special variable storing of log-
posterior

Target += - (x-1)^2;
// We can directly add anything to log-posterior
```

Vectorization

- A lot of sampling statements could be vectorized

```
for (i in 1:n)
{
   x[n] ~ normal(mu[n], sigma[n]);
}

// equivalent to

x ~ normal (mu, sigma);
// This is faster
```

Stan linear algebra

- All standard operators apply.
- Multiplication of vectors, matrices is not element-wise

```
matrix[20,20] m1;
matrix[20,20] m2;
matrix[20,20] res;

res = m1*m2 ; // matrix product
Res = m1/m2 ; // matrix division
Res = m1 * inverse(m2) //
res = m1 .* m2; // elementwise

res = m1’ ; // transposition
```

- Many matrix operations (Cholesky decompositions)

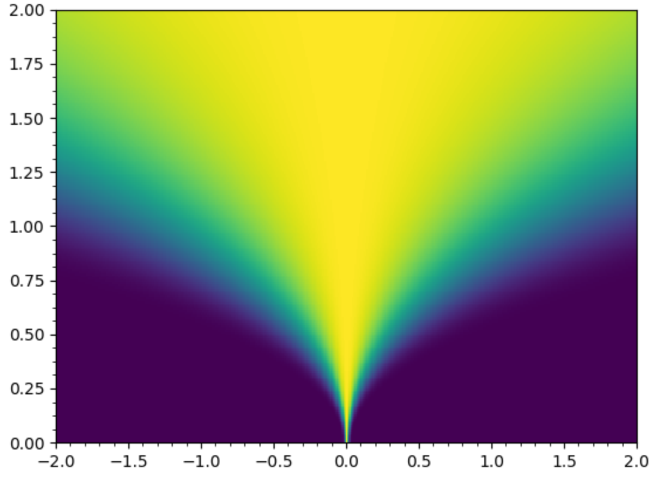
Algorithms

- Stan implements
- NUTS (the default algorithm)
- Pure Hamiltonian Monte-Carlo
- (Penalized) maximum likelihood using LM-BFGS (limited memory Broydon Fletcher Goldfarb Shannon)
- ADVI (AutoDiff Variational Inference)

Potential problems for Stan

- Multi-modality

  HMC cannot explore separated modes
- Posteriors with vastly variable curvatures (funnel distributions; see Neal 2003)
- No support for discrete parameters
- Likelihood need to be expressed in stan language
- Model comparison



CmdStan vs pystan

- Cmdstan – better for large models – the chains aren't stored in memory
- CmdStan – you need to run parallel chains by hand

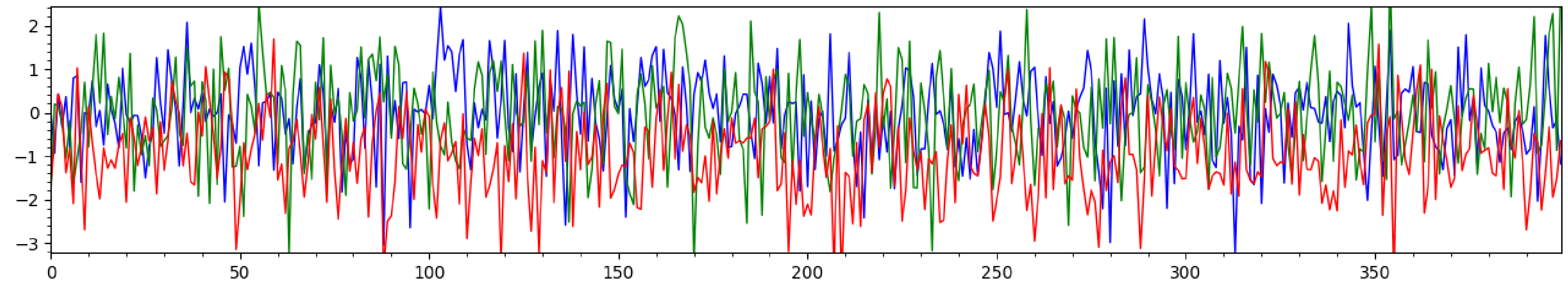
Markov-Chain Convergence checking

- Many ways to test. Stan provides two diagnostics:
- R-hat Gelman&Rubin diagnostic

$$W=\frac{1}{m}\sum s_j^2 \qquad s_j^2=\frac{1}{n-1}\sum_{i=1}^{n}(\theta_{ij}-\bar{\theta}_j)^2$$ Within the chain variance

$$B=\frac{n}{m-1}\sum_{j=1}^{m}(\bar{\theta}_j-\bar{\bar{\theta}})^2 \qquad \bar{\bar{\theta}}=\frac{1}{m}\sum_{j=1}^{m}\bar{\theta}_j$$ Between the chain variance

- $\hat{R}=\sqrt{1-\frac{1}{n}+\frac{B}{W}}$ Must be around 1 (1.01 -1.1)
-
-
-
-



- Number of Effective samples

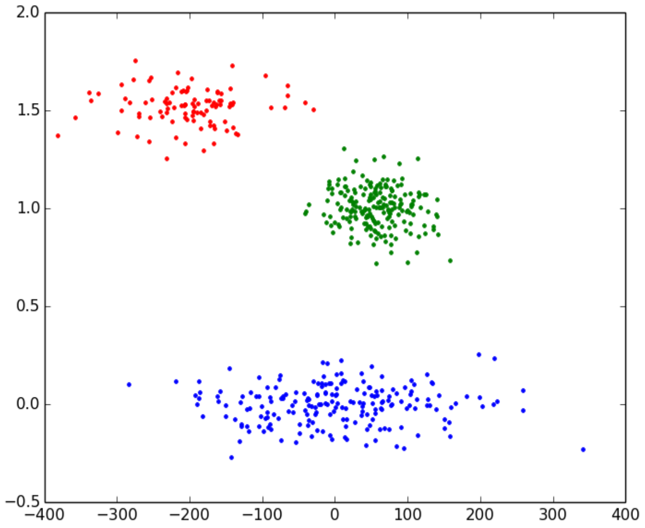
Mixture modeling

- P(x|c) the PDF for every given class
- P(x,c) is true joint PDF, but Stan doesn't support discrete parameters

$$P(x) = \int P(x,c)dc$$

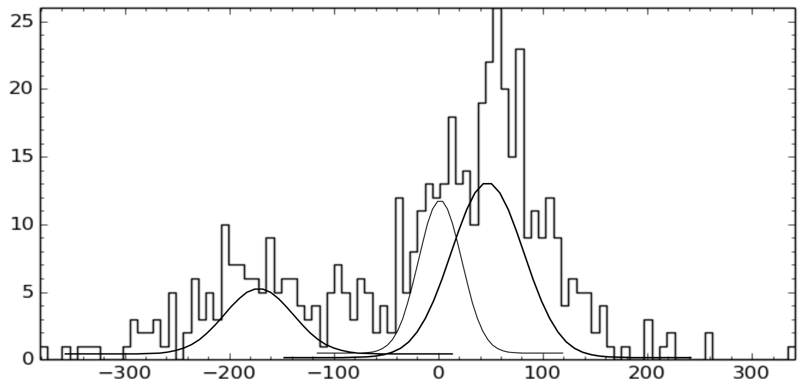
$$P(x) = \int P(x|c)P(c)dc$$

$$P(x) = \sum_i P(c_i)P(x|c_i)$$



- PDF of the mixture is the weighted sum of PDFs

$$P(x) = \sum \alpha_i P_i(x), \qquad \sum \alpha_i = 1$$



Posterior predictive checks

- Checking goodness of fit of large Bayesian models is hard
- One possibility is via Marginal likelihoods (evidences) but you **need** alternative models
- Create artificial data from posterior samples
- $P(\hat{D}|D) = \int P(\hat{D}|\phi)P(\phi|D)d\theta$
- For each replicated Dataset we can compute some statistics (i.e. percentiles) $\{F(\hat{D})\}$
  We can compare it to the data $F(D)$

Model comparison/selection in Stan

- $P(\phi|D,H_1) = \dfrac{P(D|\phi,H_1)\pi(\phi|H_1)}{P(D|H_1)}$
- Stan does not compute the marginal likelihoods
- You can compute marginal likelihoods using Laplace approximation (I.e approximate the Posterior by a Normal distribution)
- Or use the predictive properties of the models to
- Cross-validated likelihood
- Watanabe-Akaike Criterion (see Gelman+2014)

CmdStan vs pystan

- Cmdstan – better for large models – the chains aren't stored in memory
- CmdStan – you need to run parallel chains by hand

Intro into hierarchical models

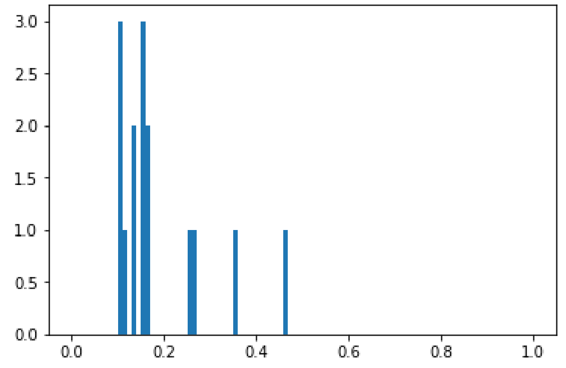
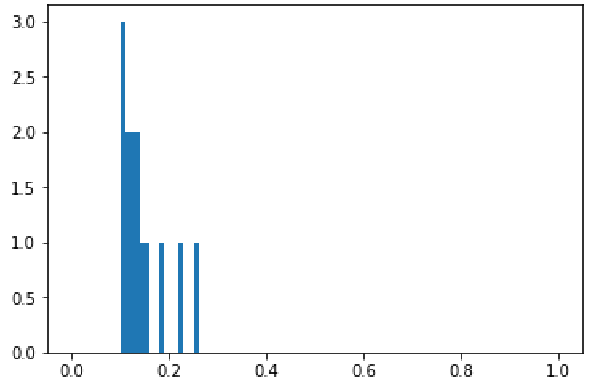
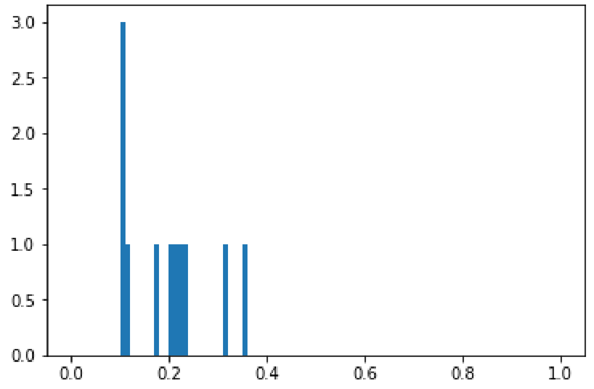
- Measurement of some property (mass of stars in clusters, luminosity of galaxies in galaxy groups…)
- Our model is a power-law (j is id of the system)

$$\mathcal{P}(x) \propto x^{\alpha}$$

$$\{x_{i,j}\} \sim Powerlaw(\alpha_j)$$

$$\alpha_j \sim U(-5,-1)$$

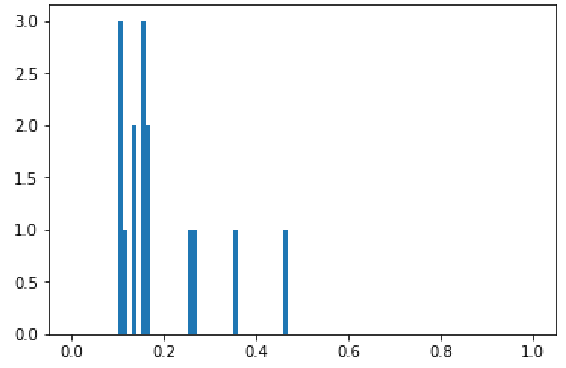
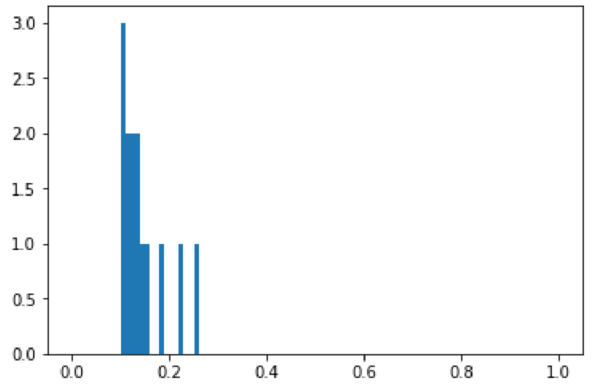
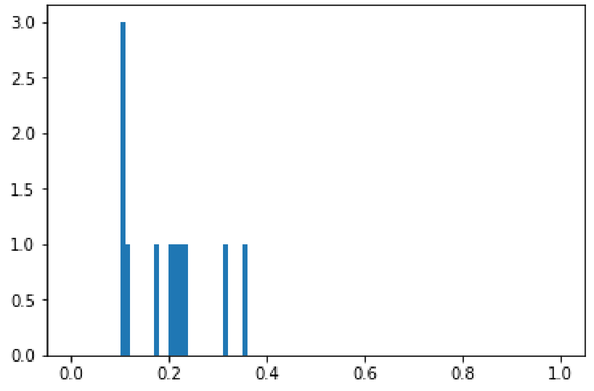
- The problem is that for every case the measurement is very noisy
- How can we understand the distibution of α's ?

Intro into hierarchical models

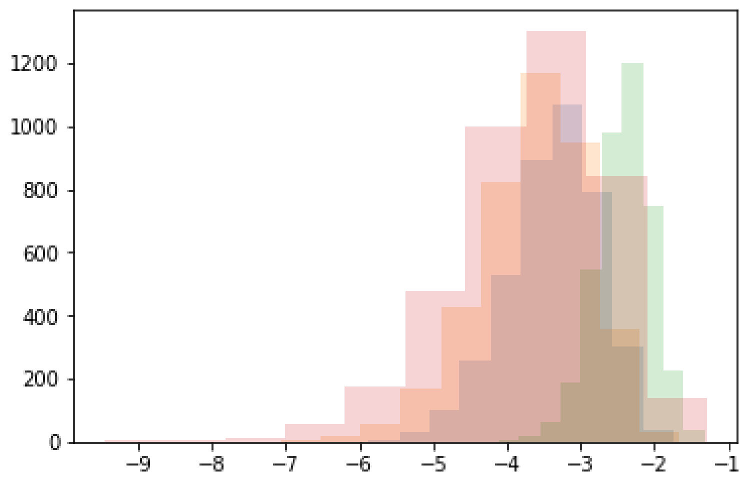
- Model $\mathcal{P}(x) \propto x^{\alpha}$
- N systems (j=1..N) with different slopes α_j
- $\{x_{i,j}\} \sim Powerlaw(\alpha_j)$
- We want to measure the average slope and the scatter of slopes
- We put a prior on the slopes
- $\alpha_j \sim \mathcal{N}(\mu, \sigma)$
- Where μ, σ are new parameters for average slope and scatter in the slopes

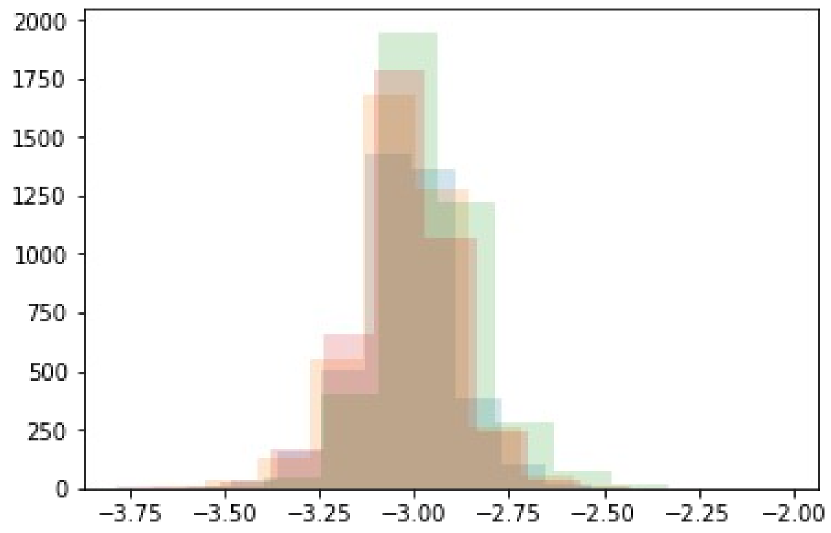
$$\mu \sim U(-5, -1)$$
$$\sigma \sim LogNormal(0, 1)$$

Shrinking in hierarchical models

- After imposing the hyper-prior the posteriors for each object shrink!





ADVI

- **Approximate** method of inference for very large models
- AutoDiff Variational Inference
- Variational Inference – Approximate the posterior
  $P(\phi|D)$
  by some function family,
  parametrised by q $F_q(\phi)$
- Sampling is replaced by minimizing KL-divergence
  $D_{KL}(P(\phi|D)|F_q(\phi))$
- Example of $F_q(\phi)$ (independent Gaussians) (called mean field approximation)

$$F_q(\phi) \propto \exp\left[-\frac{(\phi_1 - q_{\mu,1})^2}{2q_{\sigma,1}^2}\right] \exp\left[-\frac{(\phi_2 - q_{\mu,2})^2}{2q_{\sigma,2}^2}\right] \dots$$

ADVI in Stan

- Constrained variables are modeled by Gaussians in the transformed unconstrained space.
- Mean-field/full covariance mode
- Optimization is done by stochastic gradient descent in batches

Running ADVI

- CmdStan

```
$ ./power_law method=variational  data
file=distr500.data.R

Begin eta adaptation.
Iteration:   1 / 250 [  0%]  (Adaptation)
Iteration:  50 / 250 [ 20%]  (Adaptation)
Iteration: 100 / 250 [ 40%]  (Adaptation)
```

- Pystan

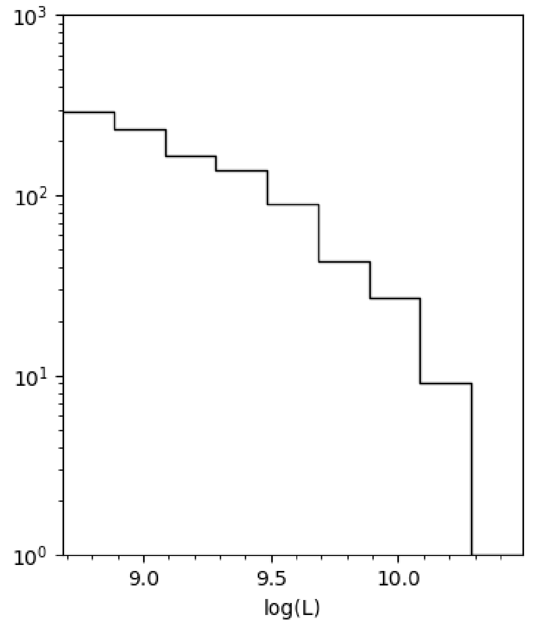
```
> mod = pystan.StanModel('gmm_advi.stan')
> mod.vb(data)
..

Begin eta adaptation.
Iteration:   1 / 250 [  0%]  (Adaptation)
Iteration:  50 / 250 [ 20%]  (Adaptation)
..
```

Example problem: Gaia

- You are given apparent magnitudes for a set of stars their uncertainties, parallaxes and their uncertainties.
- $m_i, e_i \pi_i, \sigma_{\pi,i}$
- Find out the mean absolute magnitude of the tracer and its scatter.
- Assume that the population has the exponential distribution ~exp(-r/h) in distance from the sun

problems/gaia.data.pickle
problems/gaia.data.R

Example problem: Fit Schechter distribution



- We measure luminosities of galaxies in several surveys. L_i
- Each survey observes galaxies at different redshifts z
- $P(L) \propto L^{\alpha} exp(-L/L_o)$
- Each survey has a different low luminosity limit.
- We want to measure The evolution of the power-law slope α and L0

$$\alpha = \alpha_0 + \alpha_1 \, z$$

$$L_0 = Ł_{0,0} + L_{0,1} \, z$$

problems/gaia.data.pickle
problems/gaia.data.R

Examples

- cache_code – Python code to cache compilation
- Exp2d_pixelated – Fitting 2D pixelated distribution by an exponential profile
- gmm_advi – Multi-Dimensional Gaussian Mixture model for AutoDiff Variational Inference
- Mixtures – Mixture model for linear regression with outliers
- Plummer2d – Fitting an unbinned 2D distribution by a Plummer model
- Plummer2d_err – Fitting an unbinned 2D distribution by a Plummer model, with 2D Gaussian uncertainties
- Plummer2d_func – The same with creating a new PDF function
- Posterior_predictive – Illustration how to do Posterior predictive checks with Stan
- power_law_hierarch – Fitting an hierarchical model a large number of power-law distibutions
- Regression – Linear regression (regression with uncertainties)
- Splines – example of a non-paramatric model with large number of parameters.