Bayesian Approaches to Inverse Problems in Astrophysics and Cosmology

lecture 2

Dr. Prashin Jethwa
Institut für Astrophysik

Lecture 2

- Introduction to JAX
- Writing numpyro models
- Overview of inference algorithms
- Deep-dive into the “gold-standard” inference algorithm: HMC/NUTS

By the end of this lecture you should be able to:

- Write numpyro models
- Describe three inference techniques and the differences between them:
  - Point estimation
  - Variational inference
  - MCMC
- Explain MCMC convergence diagnostics

notebook: intro_to_jax

Writing a numpyro model

- A numpyro model is a python function which implicitly defines a target probability distribution
- Build the model using familiar probability distributions which are defined in

  ```
  import numpyro.distributions as dist
  ```
- The model can contain:
  - sampling statements:

    ```
    x = numpyro.sample(`x`, dist.Normal(0, 1))
    ```
    → x is sampled from a unit normal distribution
  - deterministic transformations:

    ```
    z = numpyro.deterministic(`z`, x+y)
    ```
    → z is the (deterministic) sum of random variables x and y
  - plates to identify independent and identically distributed variables:

    ```
    numpyro.plate
    ```
- If some variable is observed (i.e. we have data for this variable) mark this using the `obs` keyword

  ```
  x = numpyro.sample(`x`, dist.Normal(0, 1), obs=x_obs)
  ```
- All calculations in the model must be compatible with JAX
  - use `jax.numpy` instead of `numpy` and `jax.scipy` instead of `scipy`

Using a numpyro model

Once you've written a model, numpyro provides functions that you can run on your model:

- `numpyro.render_model` – draw the Bayesian Network for your model
- `numpyro.initialize_model` – extract the target probability distribution function from the model
- `numpyro.infer` – a library containing many ***gradient based inference*** algorithms that you can run on your model

notebook: hierarchical_linear_regrerssion_lecture2

Overview of inference techniques

From simplest to more complex:

1. Conjugate Priors
2. Point Estimators
3. Laplace Estimators / Fisher Information Matrix
4. Variational Inference
5. Monte Carlo Sampling
6. Markov Chain Monte Carlo (MCMC) Sampling

I'll conceptually describe these inference techniques & introduce an important concept: the *typical set*.

Conjugate priors

- For some likelihoods, there is a choice of prior so that the posterior is analytic + in same family as prior, e.g.

Gaussian likelihood x Gaussian prior → Gaussian posterior
Poisson likelihood x Gamma prior → Gamma posterior
Binomial likelihood x Beta prior → Beta posterior

- Such a prior is called the conjugate prior for the given likelihood

Conjugate priors

- For some likelihoods, there is a choice of prior so that the posterior is analytic + in same family as prior, e.g.

Gaussian likelihood x Gaussian prior → Gaussian posterior
Poisson likelihood x Gamma prior → Gamma posterior
Binomial likelihood x Beta prior → Beta posterior

- Such a prior is called the conjugate prior for the given likelihood

- ***Pros***: exact calculation of posterior probabilities (+ sometimes the model evidence)
- ***Cons***: realistic models aren't described by simple likelihoods and their conjugate priors

Point estimators

Bayes' theorem:

$$p(\theta \mid y) = p(y \mid \theta)\, p(\theta) \,/\, p(y)$$

Posterior = Likelihood × Prior / Evidence

Point estimators

Bayes' theorem: $p(\theta \mid y) = p(y \mid \theta)\, p(\theta) \,/\, p(y)$

Posterior | Likelihood | Prior | Evidence

A *point estimator* finds a single acceptable value of θ. Two varieties:

- Maximum likelihood (ML) estimator:
  - the value of θ which maximises the likelihood i.e. $\theta_{ML} = \text{argmax}_{\theta}\, p(y \mid \theta)$

Point estimators

Bayes' theorem:

$$p(\theta \mid y) = p(y \mid \theta)\, p(\theta) \,/\, p(y)$$

Posterior | Likelihood | Prior | Evidence

A *point estimator* finds a single acceptable value of θ. Two varieties:

- Maximum likelihood (ML) estimator:
  - the value of θ which maximises the likelihood i.e. $\theta_{ML} = \text{argmax}_\theta\, p(y \mid \theta)$
- Maximum A Posteriori (MAP) estimator:
  - the value of θ which maximises the posterior i.e. $\theta_{MAP} = \text{argmax}_\theta\, p(\theta \mid y)$

ML vs. MAP. Or... why do we need a prior?

- Where does the prior come from?

- Our *domain-knowledge*
  - previous measurements, results from the literature etc...
  - order of magnitude estimates
  - physical constraints e.g. masses > 0
  - desirable properties e.g. a result should be *smooth*

- But the likelihood comes from the data! Won't a prior bias our results?
  - Sometimes no: if we have enough data, then the likelihood will outweigh the prior
  - Sometimes yes! And that's OK.

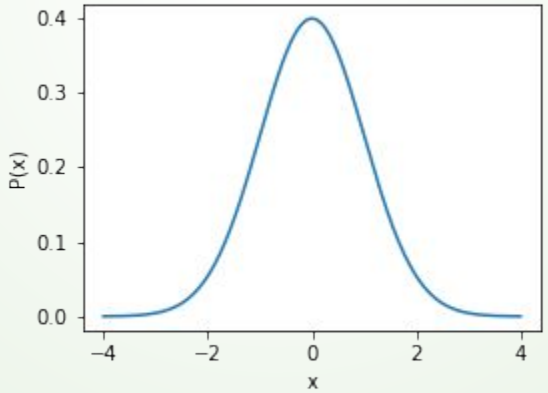
- However, any single point estimate (ML or MAP) may be very unrepresentative of the majority of solutions

Example: length of normal vectors

Say we sample a *single* value from a unit-normal

x ~ Normal(0 , 1)

What is the most likely value of |x|?

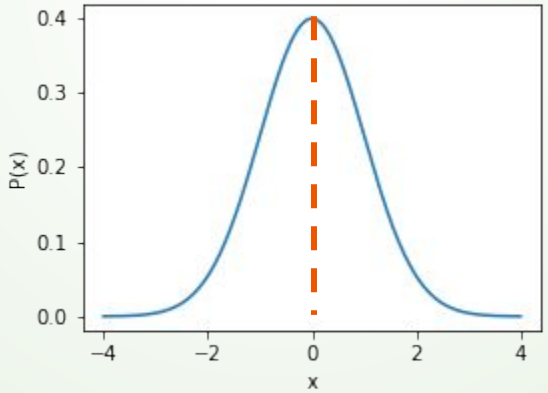


Example: length of normal vectors

Say we sample a *single* value from a unit-normal

$x \sim \text{Normal}(0, 1)$

What is the most likely value of $|x|$?



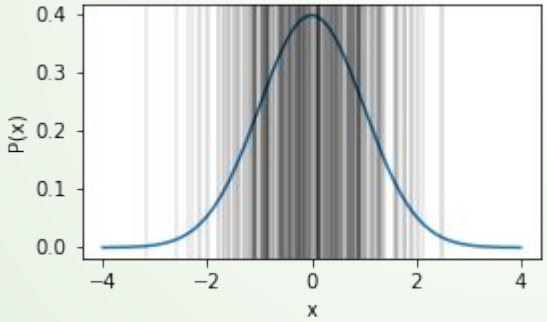
0 is the most likely value of $|x|$

Example: length of normal vectors

Now, say we sample *10,000* values from a unit-normal

x ~ Normal(0 , 1)

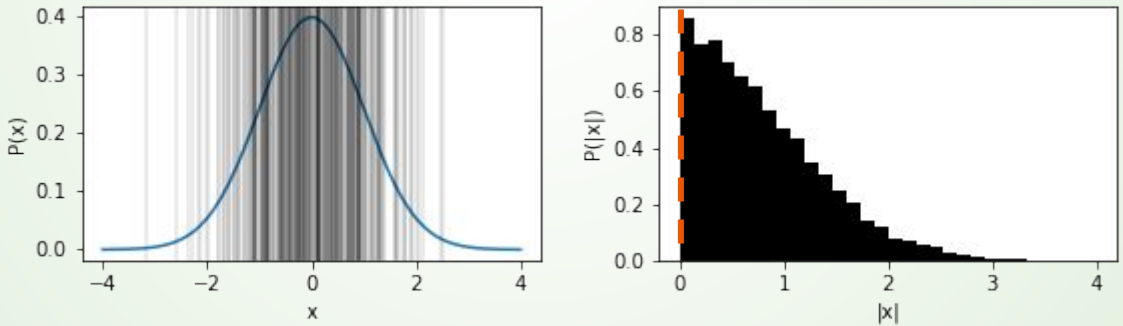
What is the most likely value of |x|?



Example: length of normal vectors

Now, say we sample *10,000* values from a unit-normal
x ~ Normal(0 , 1)

What is the most likely value of |x|?



histogram of |x| from samples

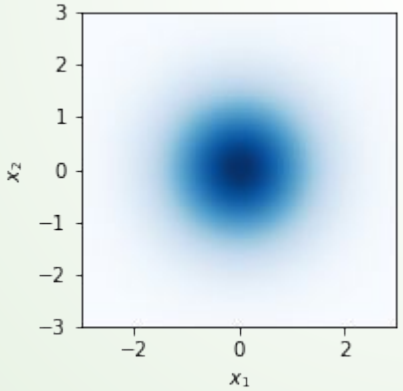
0 is the most likely value of |x|

Example: length of normal vectors

Now, say we sample a *single* point from a ***2D*** unit- multivariate normal

$$x \sim \text{Normal}(\mathbf{0}, I_2)$$

What is the most likely value of $|x|$?



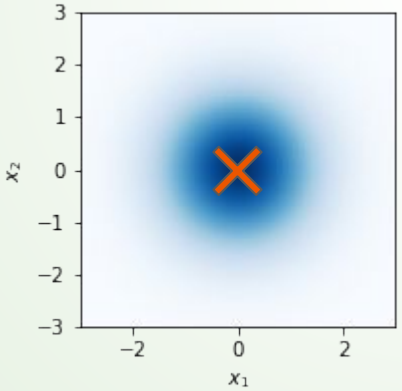
Example: length of normal vectors

Now, say we sample a *single* point from a ***2D*** unit- multivariate normal

$$x \sim \text{Normal}(\mathbf{0}, I_2)$$

What is the most likely value of $|x|$?

0 is the most likely value of $|x|$

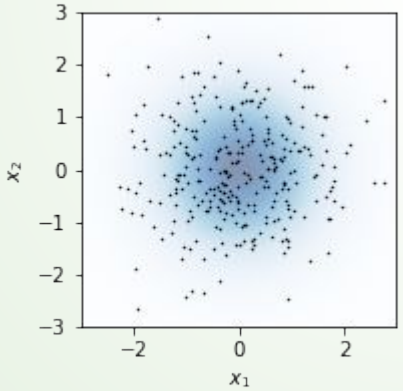


Example: length of normal vectors

Now, say we sample *10,000* values from a ***2D*** unit- multivariate normal

$$x \sim \text{Normal}(\mathbf{0}, I_2)$$

What is the most likely value of $|x|$?

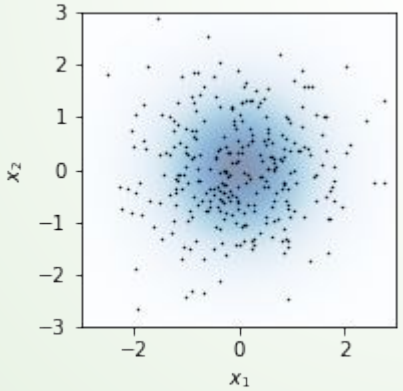


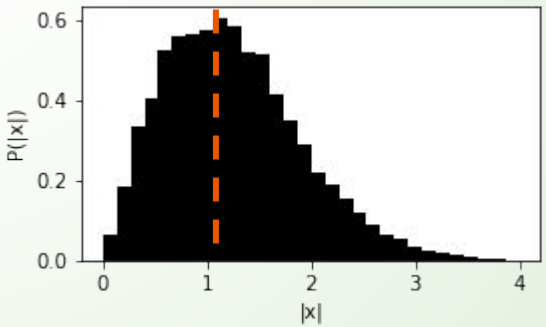
Example: length of normal vectors

Now, say we sample *10,000* values from a ***2D*** unit- multivariate normal

$$x \sim \text{Normal}(\mathbf{0}, I_2)$$

What is the most likely value of |x|?



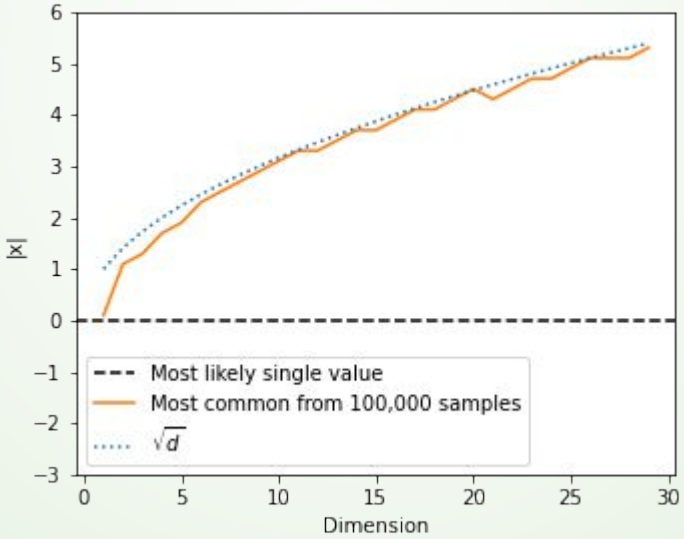


histogram of |x| from samples

The most common value of |x| is 1, not 0

Example: length of normal vectors

As we increase dimension, the difference between the most likely *single* value of |x| and the most common value amongst samples grows and grows:

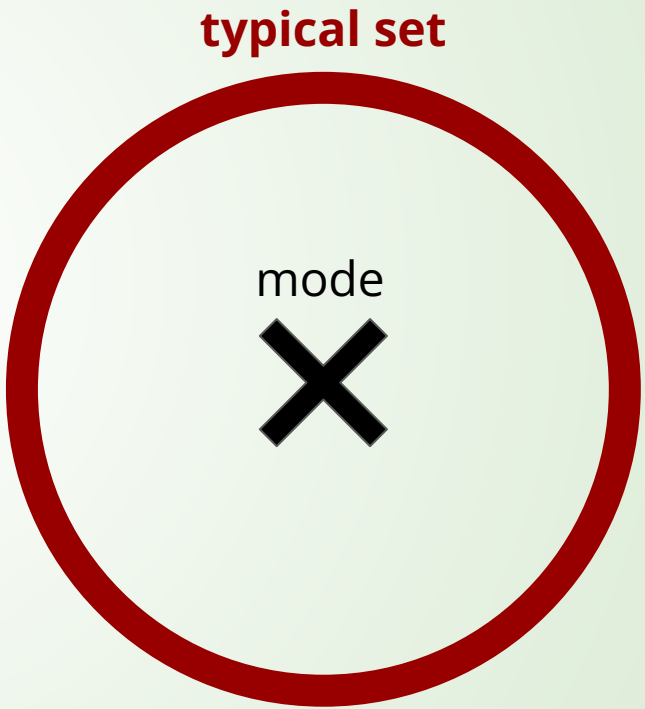


Point estimates may be bad in high dimensions

- The mode can be unrepresentative of the majority of the distribution. Why?
- probability mass = probability density x volume
  - volume grows exponentially as dimension increases
  - volume around the mode is small
  - *typical solutions* move away from the mode in high dimension

- Useful concept: the *typical set* of a distribution
  - the region where samples tend to lie
  - in high-dimensions, typical set forms a doughnut/shell surrounding the mode

- Inference should ideally characterise the typical set, not just the mode



Overview of inference techniques

From simplest to more complex:

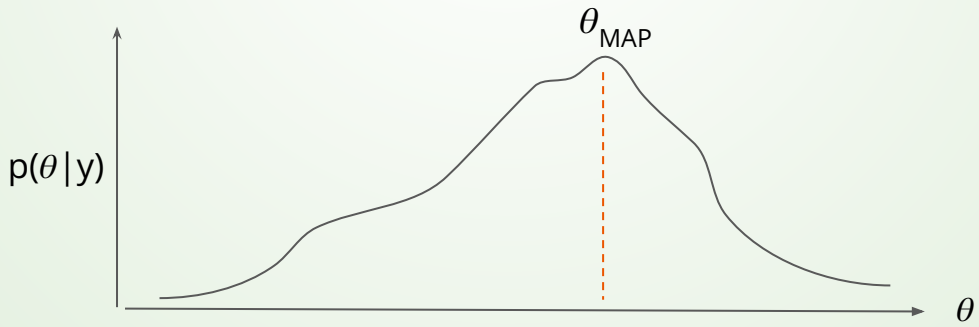
1. Conjugate Priors
2. Point Estimators
3. Laplace Estimator / Fisher Information Matrix
4. Variational Inference
5. Monte Carlo Sampling
6. Markov Chain Monte Carlo (MCMC) Sampling

These methods characterise the *typical set*, not just a point-estimate

I'll conceptually describe these 5 types of inference techniques & introduce an important concept for inference: the *typical set*.

Laplace Approximation

1. Find the MAP solution θ_{MAP}

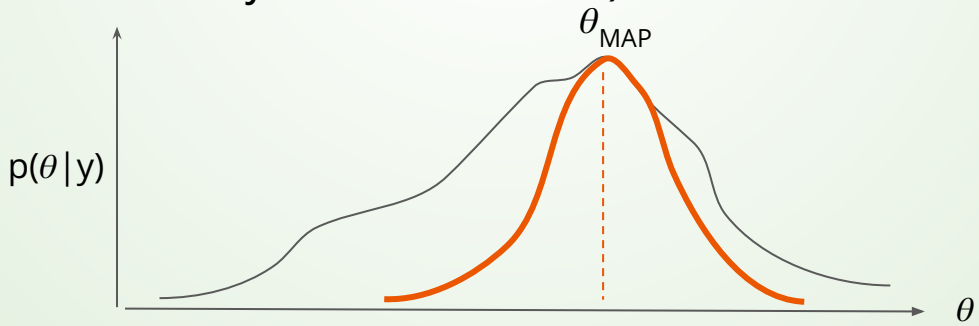


Laplace Approximation

1. Find the MAP solution θ_{MAP}
2. At the MAP solution, calculate the curvature of the posterior

   Given by the Hessian of log posterior with respect to model parameters i.e. matrix H with elements $h_{ij} = \frac{\partial^2 \log P(\theta \mid y)}{\partial\theta_i \, \partial\theta_j}\Big|_{\theta = \theta_{MAP}}$

3. Approximate the posterior as normal with mean θ_{MAP} and covariance matrix $-H^{-1}$ (the *Fisher Information Matrix)*



Laplace Approximation

1. Find the MAP solution θ_{MAP}
2. At the MAP solution, calculate the curvature of the posterior:

   Given by the Hessian of log posterior with respect to model parameters i.e. matrix H with elements $h_{ij} = \frac{\partial^2 \log P(\theta \mid y)}{\partial\theta_i \, \partial\theta_j}\Big|_{\theta = \theta_{MAP}}$

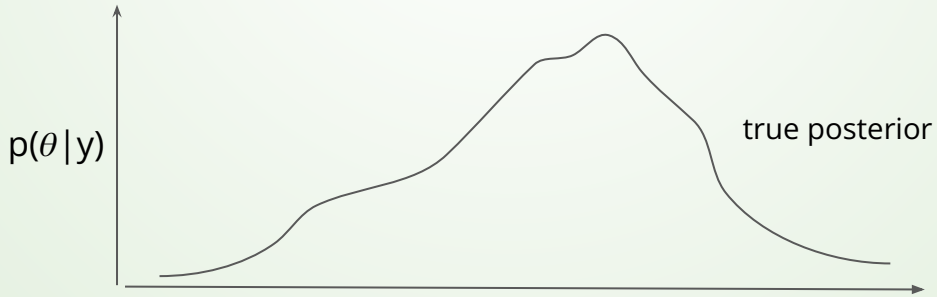
3. Approximate the posterior as normal with mean θ_{MAP} and covariance matrix $-H^{-1}$ (the *Fisher Information Matrix)*

Pros: fast, easy

Cons: posterior often not well-characterised by curvature at the mode

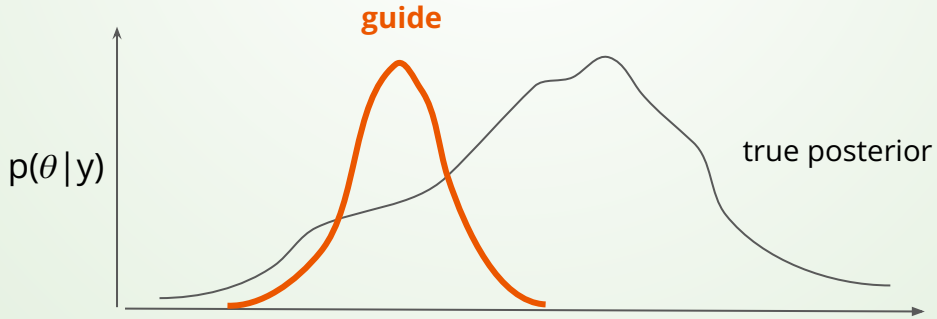
Variational Inference

- Approximate the posterior with a guide distribution with some parameters - variational parameters, ϕ
- Find the optimum ϕ_* such that the guide best matches the true posterior



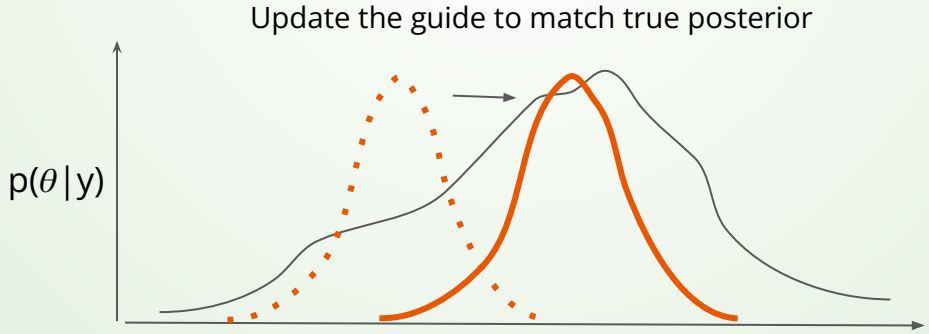
Variational Inference

- Approximate the posterior with a guide distribution with some parameters - variational parameters, ϕ
- Find the optimum ϕ_* such that the guide best matches the true posterior
- e.g. if the guide is a Gaussian:



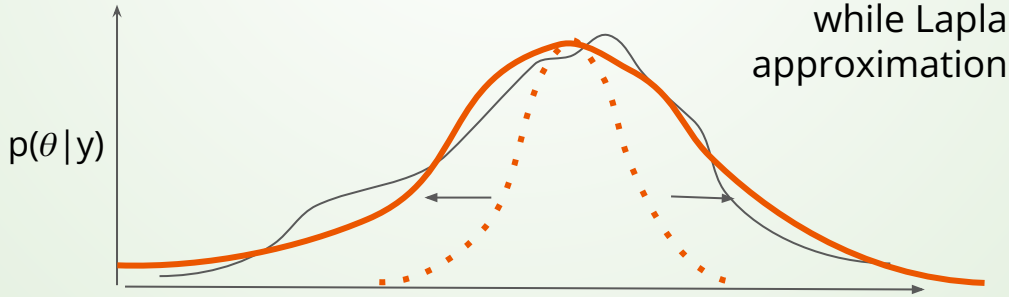
Variational Inference

- Approximate the posterior with a guide distribution with some parameters - variational parameters, ϕ
- Find the optimum ϕ_* such that the guide best matches the true posterior
- e.g. if the guide is a Gaussian:



Variational Inference

- Approximate the posterior with a guide distribution with some parameters - variational parameters, ϕ
- Find the optimum ϕ_* such that the guide best matches the true posterior
- e.g. if the guide is a Gaussian:



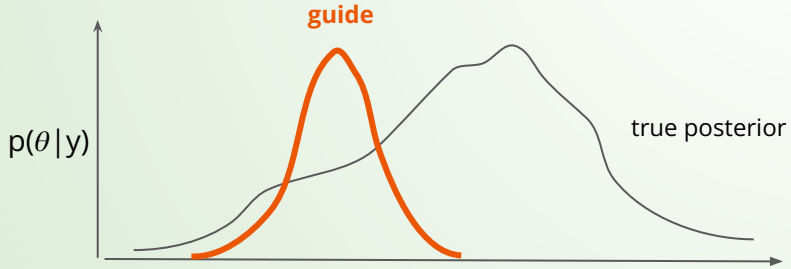
This finds the best Gaussian approximation to the *whole* posterior, while Laplace finds a Gaussian approximation only valid at the mode

Variational Inference

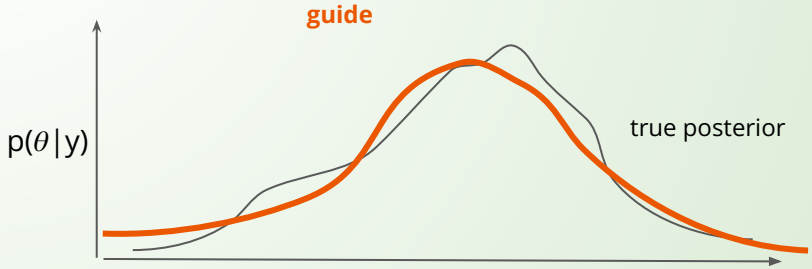
- Approximate the posterior with a guide distribution with some parameters - variational parameters, ϕ
- Find the optimum ϕ_* such that the guide best matches the true posterior
- Gaussian is just an example: there are many types of guide

Variational Inference

- Approximate the posterior with a guide distribution with some parameters - variational parameters, ϕ
- Find the optimum ϕ_* such that the guide best matches the true posterior
- Gaussian is just an example: there are many types of guide
- The difference between between the true posterior and guide is estimated using a metric called the Evidence Lower Bound = ELBO



small ELBO



large ELBO

Variational Inference

- Approximate the posterior with a guide distribution with some parameters - variational parameters, ϕ
- Find the optimum ϕ_* such that the guide best matches the true posterior
- Gaussian is just an example: there are many types of guide
- The difference between between the true posterior and guide is estimated using a metric called the Evidence Lower Bound = ELBO
- Guide chosen so that it can be easily sampled from

Variational Inference

- Approximate the posterior with a guide distribution with some parameters - variational parameters, ϕ
- Find the optimum ϕ_* such that the guide best matches the true posterior
- Gaussian is just an example: there are many types of guide
- The difference between between the true posterior and guide is estimated using a metric called the Evidence Lower Bound = ELBO
- Guide chosen so that it can be easily sampled from
- **Pros:**
  - characterises the whole posterior through optimisation
- **Cons:**
  - the true posterior might not be well described by the guide

Monte-Carlo sampling

- Algorithms to draw *exact* samples for a handful of distributions
- Samples (by definition) trace the typical set $\theta_i \sim p(\theta \mid y)$
- If we can sample from the posterior, we can estimate the value of any function $f(\theta)$:

$$E_{p(\theta|y)}[f] = \int p(\theta|y)\, f(\theta)\, d\theta \approx 1/N \sum_{i=1,\ldots,N} f(\theta_i)$$

  - Uncertainty of this estimate is given by the Monte Carlo Standard Error (MC-SE):

$$MC\text{-}SE_N[f] = StdDev(f(\theta_i)) / \sqrt{N}$$

i.e. to get more accurate estimates, you need more samples

Markov-Chain Monte Carlo (MCMC) Sampling

- Exact Monte Carlo sampling only possibly for a handful of simple distributions
- MCMC is an approximate sampling method where samples are no longer independent
- instead samples form a Markov chain i.e. sample $\boldsymbol{\theta}_{m+1}$ can depend on $\boldsymbol{\theta}_{m}$
- the dependence is encoded in transition operator

$T(\boldsymbol{\theta}_m, \boldsymbol{\theta}_{m+1})$ i.e. the probability of transitioning from $\boldsymbol{\theta}_{m} \rightarrow \boldsymbol{\theta}_{m+1}$

- If T satisfies the *detailed balance equation*, i.e.

$$p(\boldsymbol{\theta}_m)\, T(\boldsymbol{\theta}_m, \boldsymbol{\theta}_{m+1}) = p(\boldsymbol{\theta}_{m+1})\, T(\boldsymbol{\theta}_{m+1}, \boldsymbol{\theta}_m)$$

  then samples drawn according to T will converge towards the distribution $p(\boldsymbol{\theta})$
- MCMC algorithms use general purpose transition operators which can sample any distribution $P(\boldsymbol{\theta})$

Markov-Chain Monte Carlo Sampling

- samples are no longer independent
- we can define an effective number of samples N_{eff} based on autocorrelation in the Markov chain
- the Monte Carlo Standard Error:

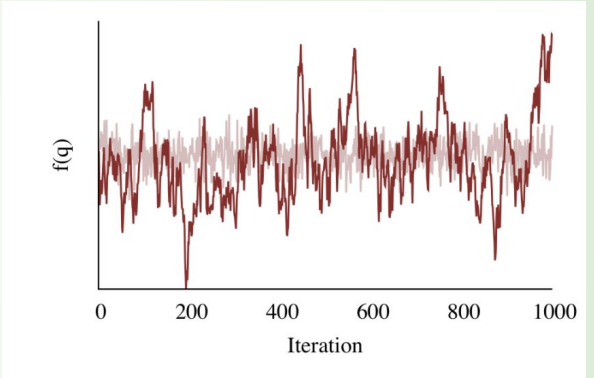
$$MC\text{-}SE_N[\,f\,] = StdDev(\,f(\,\theta_i\,)\,) \,/\sqrt{N}$$

now becomes the MCMC standard error:

$$MCMC\text{-}SE_N[\,f\,] = StdDev(\,f(\,\theta_i\,)\,) \,/\sqrt{N_{eff}}$$

- for reliable estimates we need large N_{eff}
  - i.e. many uncorrelated samples

Examples of two MCMC chains with different autocorreclations



Dark: high autocorrelation -
chains remember where they have been

Light: low autocorrelation -
chains quickly forget where they have been

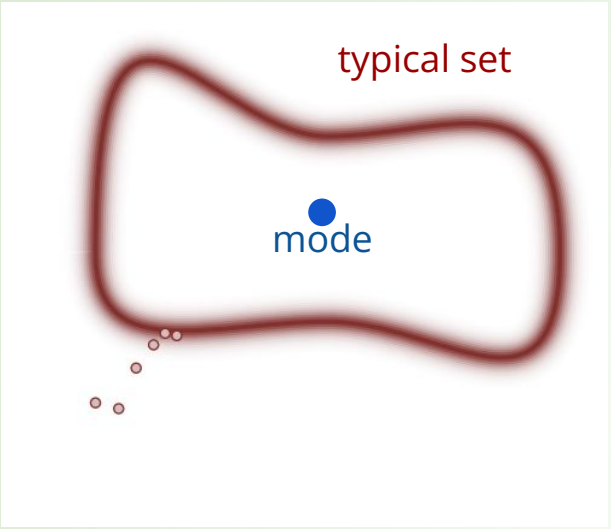
Markov-Chain Monte Carlo Sampling

- There are several different types of MCMC
  - with a different transition kernel $T(\boldsymbol{\theta}_m, \boldsymbol{\theta}_{m+1})$
  - online demonstration

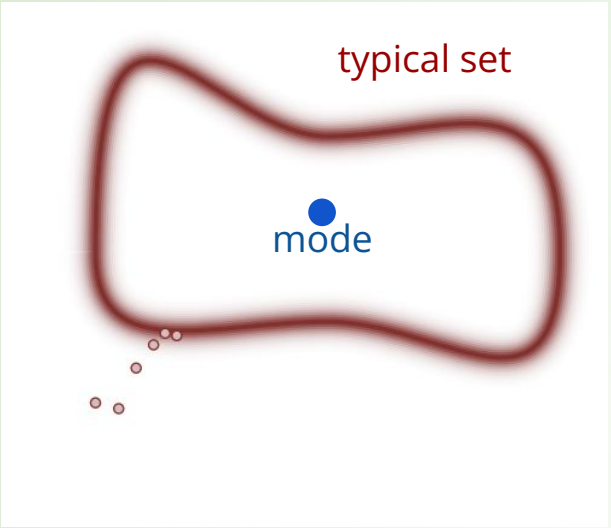
- Compare two:
  - Random Walk Metropolis Hastings
    - take a random step in a Gaussian ball around your current position
    - stochastic acceptance of proposed point based on probability density ratio
  - Hamiltonian Monte Carlo (HMC) + No-U-Turn Sampler (NUTS)
    - use Hamiltonian dynamics to *calculate trajectories through the probability distribution*
    - → higher acceptance probability + lower auto-correlation in the chains
    - Requires gradient information to calculate trajectories

Hamiltonian Monte Carlo (HMC)
&
No U-Turn Sampler (NUTS)

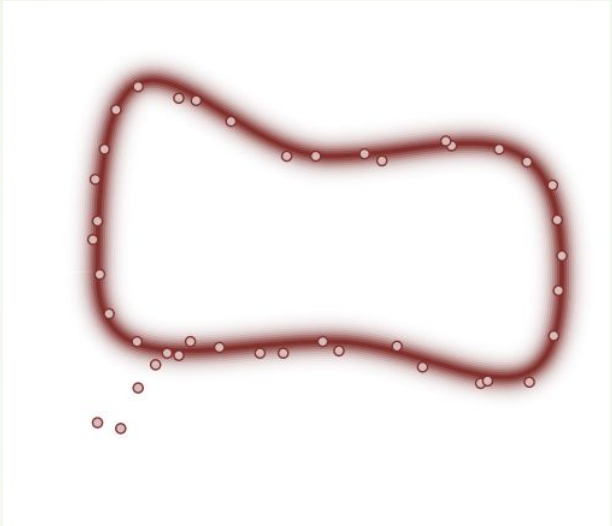
The gold-standard inference algorithm

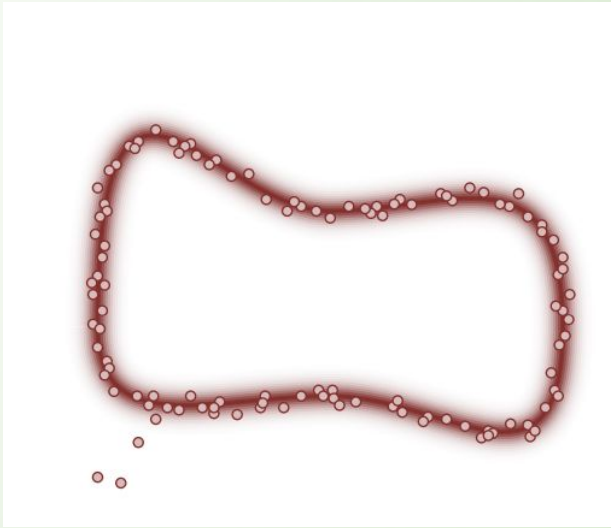


Target distribution: $p(\mathbf{x})$

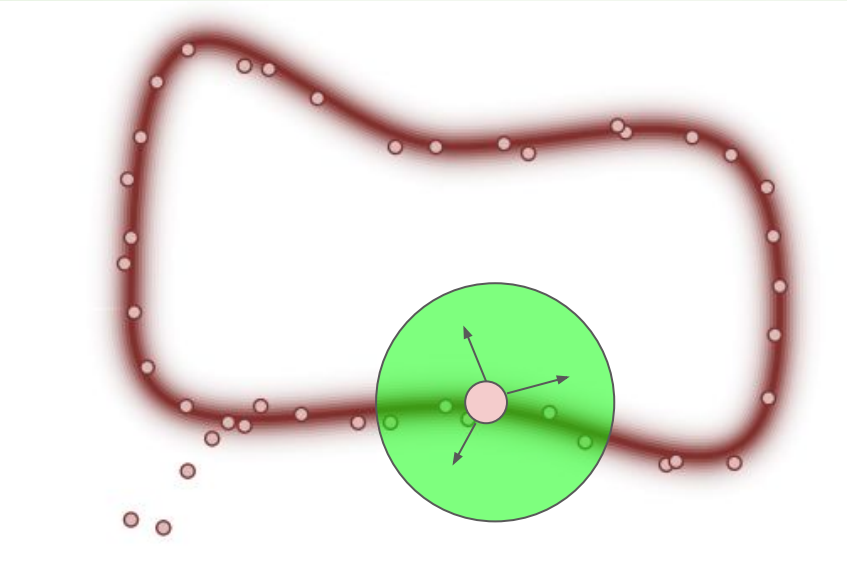


Target distribution: $p(\mathbf{x})$





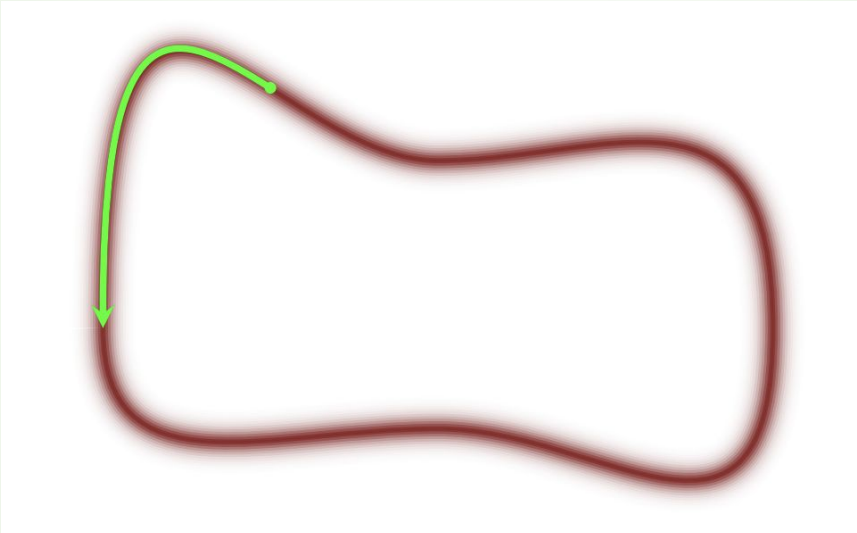
Goal: construct a MCMC transition operator that will draw many uncorrelated samples from the typical set



Random Walk Metropolis Hastings

proposes a transition to a Gaussian blob centered around current position.
This becomes very inefficient for high dimensions

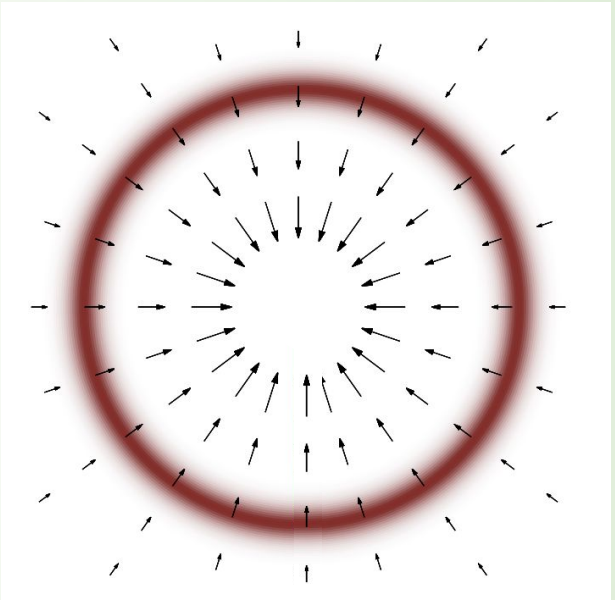
Random Walk Metropolis-Hastings
does not scale well to high dimensions.



Solution: use a transition which moves *through* the typical set towards new, unexplored neighborhoods.

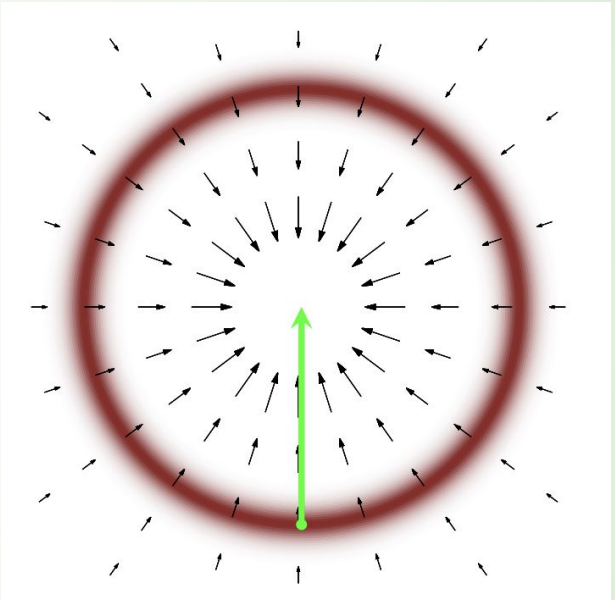
How to transition along the typical set

- Say we have target distribution $p(\mathbf{x})$
- The gradient of the target distribution $\nabla p(\mathbf{x})$ defines a vector field
- What would happen to a particle which follows a trajectory along the gradient?



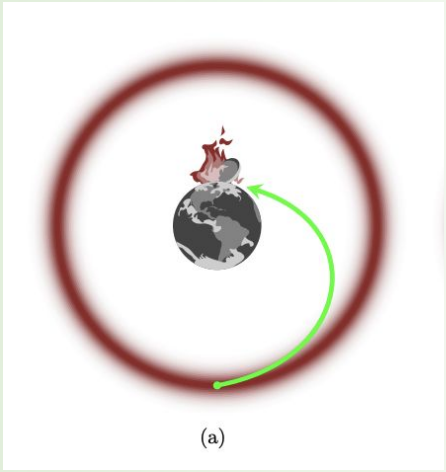
How to transition along the typical set

- Say we have target distribution $p(\mathbf{x})$
- The gradient of the target distribution $\nabla p(\mathbf{x})$ defines a vector field
- What would happen to a particle which follows a trajectory along the gradient?
  - It will end up at the mode

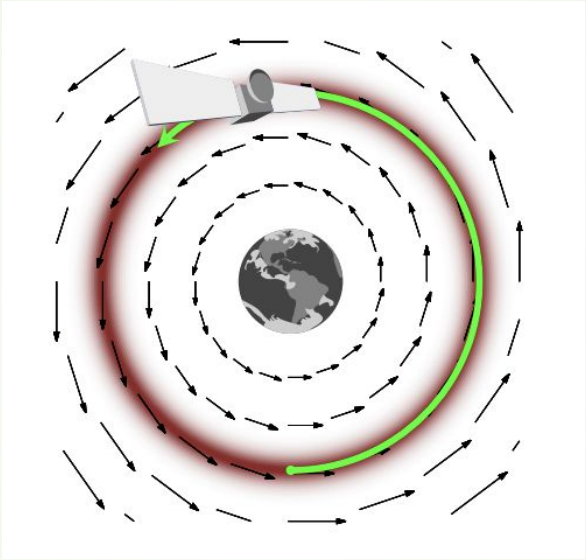


- … to instead move *along* the typical set, we must give the particles some *momentum*

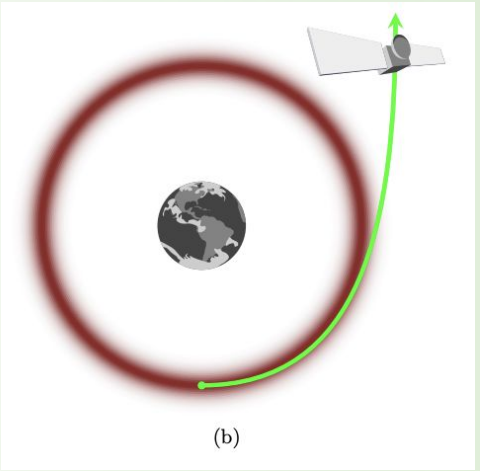
Momentum: physical analogy with a gravitational potential



Too little momentum
→ particle crashes to the mode



Correct momentum
→ particle orbits along the typical set



Too much momentum
→ particle escapes

The *correct* momentum via Hamiltonian Dynamics

- Introduce extra momentum variables $x \rightarrow (x, v)$
- Define the joint distribution over position and momentum: $p(x, v) = p(v \mid x)\, p(x)$
- Write this in terms of a Hamiltonian function H $= \exp[-H(x, v)]$

$$\rightarrow \quad H(x, v) = -\log p(v \mid x) - \log p(x)$$
$$:= K(v, x) + V(x)$$
$$\text{energy} := \text{kinetic energy} + \text{potential energy}$$

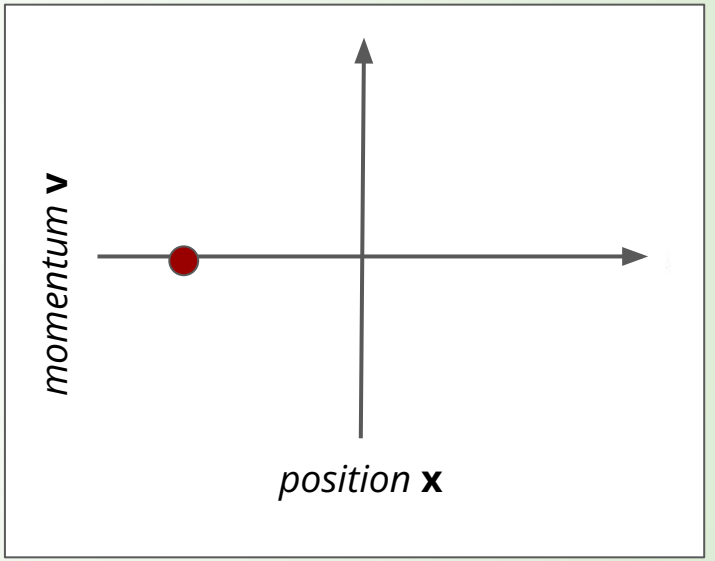
- If we take any initial point (x_0, v_0) and evolve it via Hamilton's equations:

$$dx/dt = +\partial H/\partial v$$
$$dv/dt = -\partial H/\partial x$$

then the resulting trajectory naturally orbits along the typical set of the target distribution

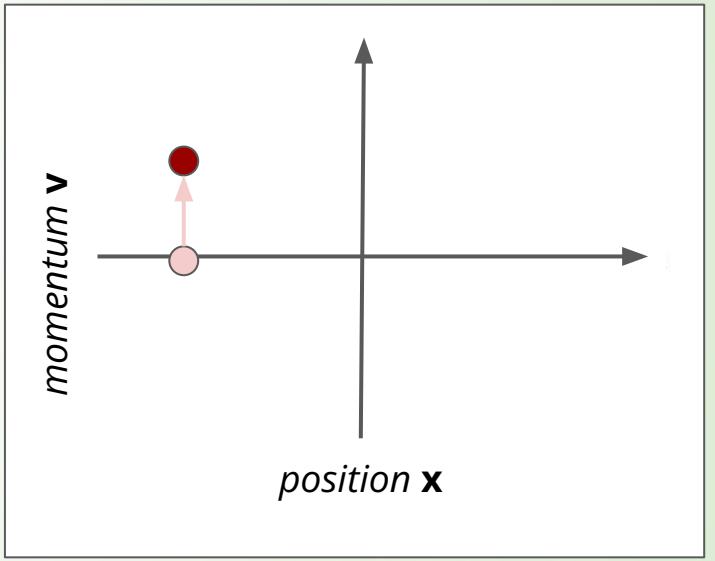
HMC: the algorithm

1. Start at initial point x_n



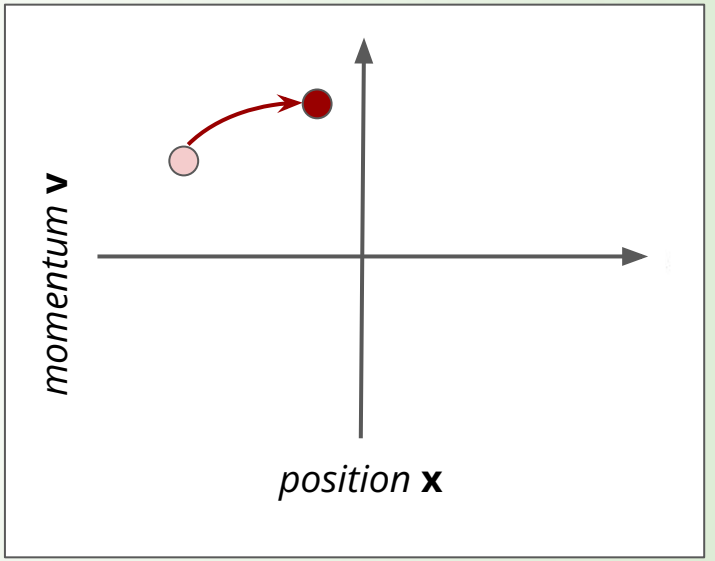
HMC: the algorithm

1. Start at initial point x_n
2. Sample a momentum $v_n \sim p(v \mid x_n)$



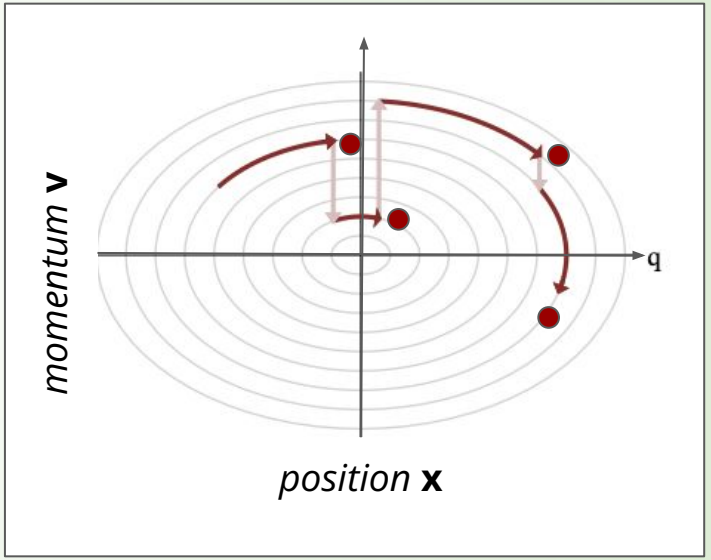
HMC: the algorithm

1. Start at initial point x_n
2. Sample a momentum $v_n \sim p(v \mid x_n)$
3. Use Hamilton's equations to numerically an orbit $(x_n, v_n) \rightarrow (x_{n+1}, v_{n+1})$



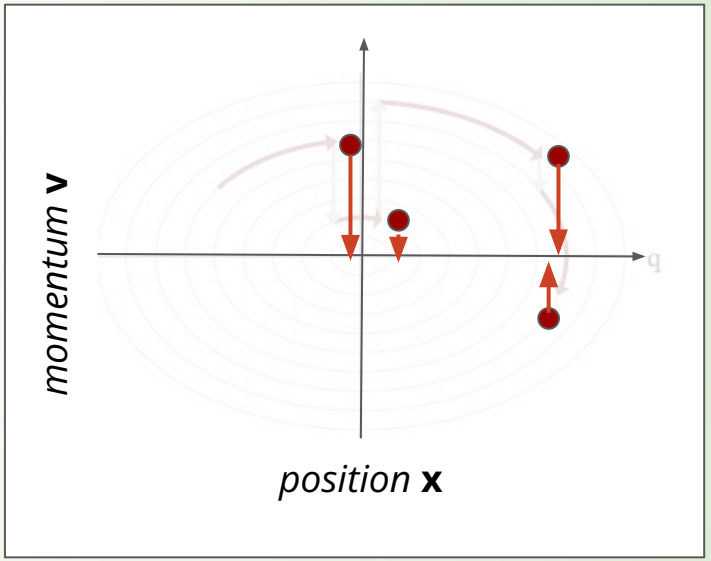
HMC: the algorithm

1. Start at initial point x_n
2. Sample a momentum $v_n \sim p(v \mid x_n)$
3. Use Hamilton's equations to numerically an orbit $(x_n, v_n) \rightarrow (x_{n+1}, v_{n+1})$
4. Accept/reject the new point…
5. Repeat for desired number of samples



HMC: the algorithm

1. Start at initial point x_n
2. Sample a momentum $v_n \sim p(v \mid x_n)$
3. Use Hamilton's equations to numerically an orbit $(x_n, v_n) \rightarrow (x_{n+1}, v_{n+1})$
4. Accept/reject the new point...
5. Repeat for desired number of samples
6. Project samples down to position space i.e. forget the momenta $(x,v) \rightarrow x$



What is $p(v \mid x)$?

- The choice of $p(v \mid x)$ - i.e. the kinetic energy term in the Hamiltonian - is a free choice - different possibilities exist
- Most common choice: Euclidean-Gaussian Kinetic Energy:

$$p(v \mid x) = N(v \mid 0 , M)$$

i.e. the kinetic energy $K(v, x) = - \log p(v \mid x)$

$$= \tfrac{1}{2} v^T M v + \log |M|$$

where M is the mass-matrix, i.e. tuning-parameters of the algorithm

- M is tuned to re-scale/rotate the parameter space distribution into a more standardised frame

Numerically integrating the orbit?

- Take an initial point (x0, v0) and evolve it via Hamilton's equations:

$$dx/dt = +\,\partial H/\partial v$$
$$dv/dt = -\,\partial H/\partial x := a$$

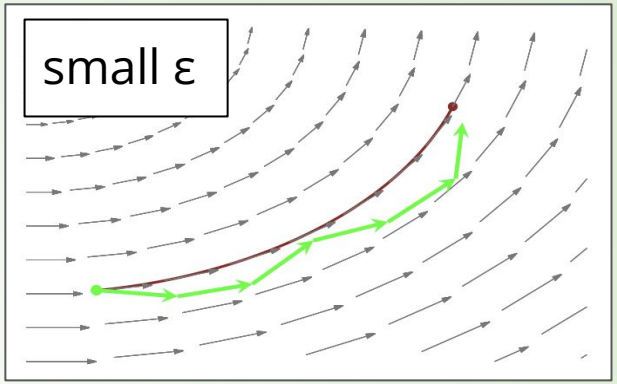
- When discretized, with some step-size **ε** these can be integrated numerically with a leapfrog (AKA kick-drift-kick) integrator:

$$v_{i+½} = v_i + a_i\,\varepsilon/2 \quad \text{“kick”}$$
$$x_{i+1} = x_i + v_{i+½}\,\varepsilon \quad \text{“drift”}$$
$$v_{i+1} = v_{i+½} + a_{i+1}\,\varepsilon/2 \quad \text{“kick”}$$

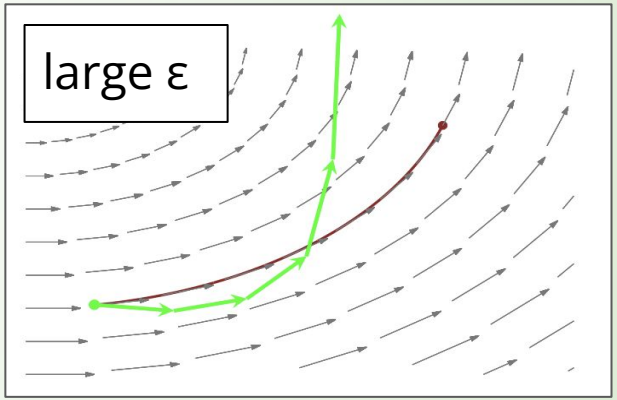
- This integrator is symplectic - i.e. it preserves energy - in contrast to other schemes e.g. Runga-Kutta methods

Setting the step-size ε

- To efficiently make large trajectories, we want a large step-size ε
- But if ε is too large, energy is no longer conserved
  - the trajectory diverges from the true one



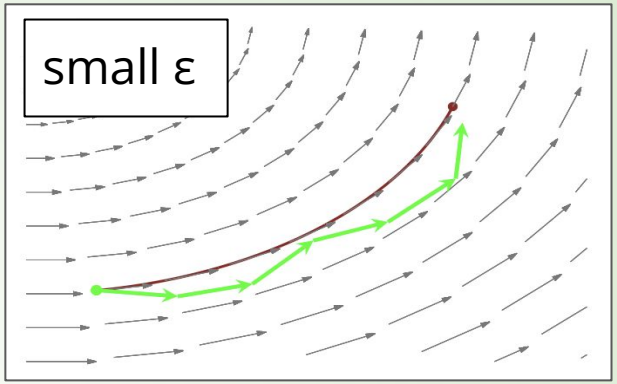
Numerical orbit stays near truth



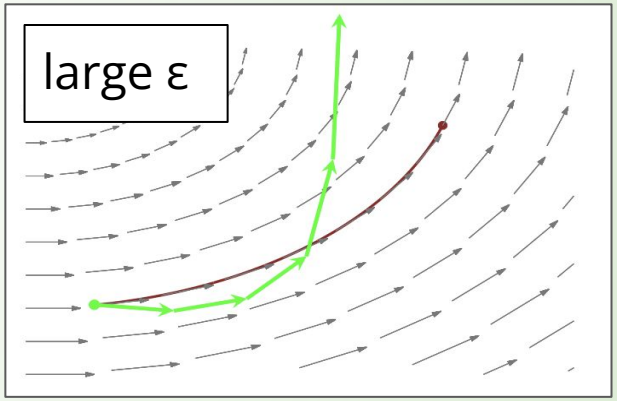
Numerical orbit diverges from truth

Setting the step-size ε

- To efficiently make large trajectories, we want a large step-size ε
- But if ε is too large, energy is no longer conserved
  - the trajectory diverges from the true one
- We can identify these cases by looking at energy drift between the trajectory endpoints:
  - $\Delta E = H(x_0, v_0) - H(x_n, v_n)$
  - If ΔE > some threshold, the trajectory is divergent
- Gives an extra step in HMC algorithm…



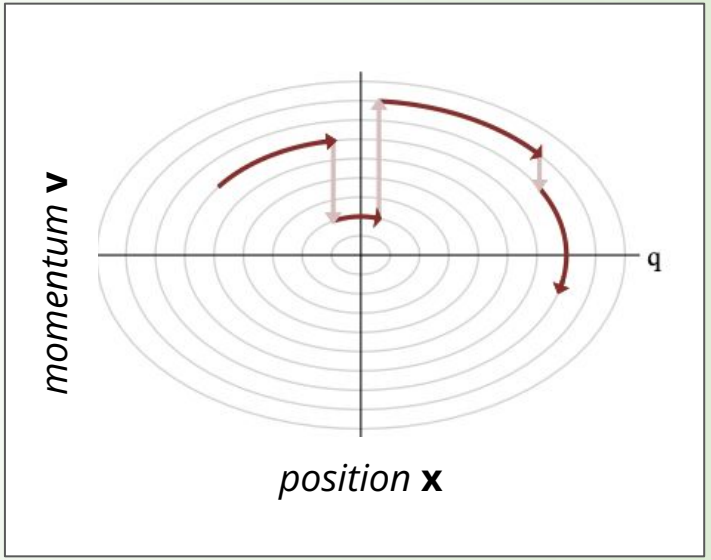
Numerical orbit stays near truth



Numerical orbit diverges from truth

HMC: the algorithm

1. Start at initial point x_n
2. Sample a momentum $v_n \sim p(v \mid x_n)$
3. Use Hamilton's equations to numerically an orbit $(x_n, v_n) \rightarrow (x_{n+1}, v_{n+1})$
4. Accept/reject new point based on energy drift in trajectory:
   a. Large drift → less likely to be accepted
   b. Implemented in a way which satisfies the detailed-balance equation
5. Repeat for desired number of samples
6. Project samples down to position space i.e. forget the momenta $(x,v) \rightarrow x$

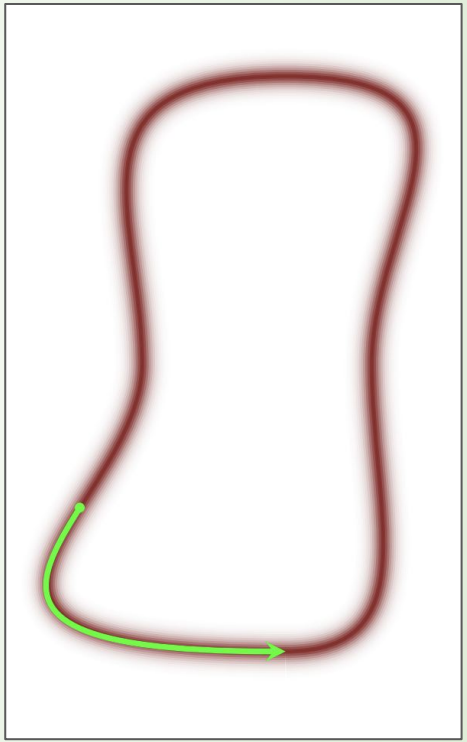


No U-Turn Sampler (NUTS)

- As well as a step-size, we need to know *how many* steps to take

No U-Turn Sampler (NUTS)

- As well as a step-size, we need to know *how many* steps to take
- We want enough steps to take us away from our initial point

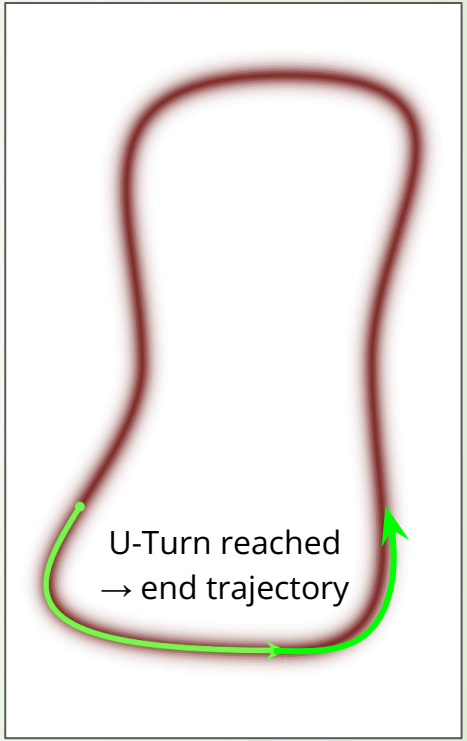


No U-Turn Sampler (NUTS)

- As well as a step-size, we need to know *how many* steps to take
- We want enough steps to take us away from our initial point
- But not so many that we return there

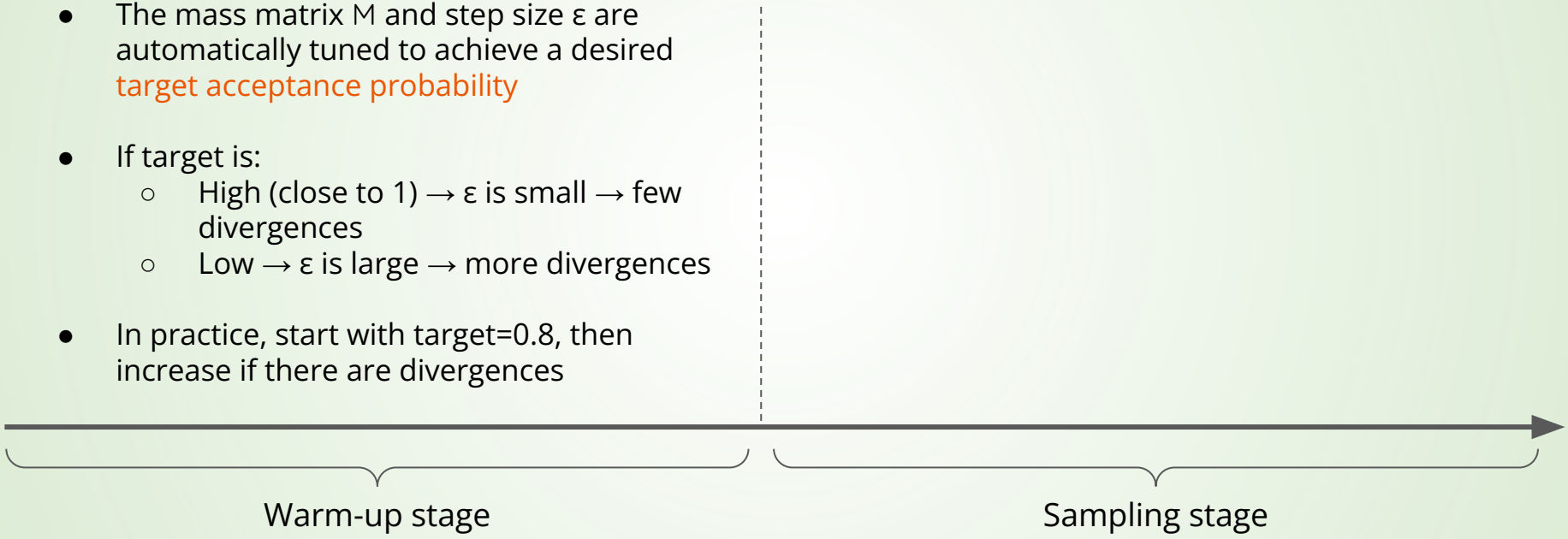
No U-Turn Sampler (NUTS)

- As well as a step-size, we need to know *how many* steps to take
- We want enough steps to take us away from our initial point
- But not so many that we return there
- Solution:
  - Implement a criteria that the trajectories should not perform a “U-Turn”
  - i.e. once they face the opposite direction, stop
  - NUTS was an important ingredient to make HMC work in practice for a wide-variety of problems



Running HMC/NUTS in practice

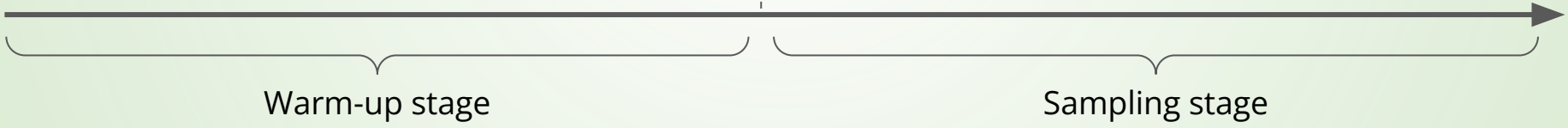
- The mass matrix M and step size ε are automatically tuned to achieve a desired target acceptance probability
- If target is:
  - High (close to 1) → ε is small → few divergences
  - Low → ε is large → more divergences
- In practice, start with target=0.8, then increase if there are divergences



Running HMC/NUTS in practice

- The mass matrix M and step size ε are automatically tuned to achieve a desired target acceptance probability
- If target is:
  - High (close to 1) → ε is small → few divergences
  - Low → ε is large → more divergences
- In practice, start with target=0.8, then increase if there are divergences

- Sampling with HMC/NUTS
  - Uses the tuned mass matrix M and step size ε from warm-up
  - Only samples from this stage are used for inference



Number of Steps in MCMC chain

Running HMC/NUTS in practice

- Posterior geometry
  - The shape (e.g. curvature) of the posterior density
  - This may change in different regions of parameter space

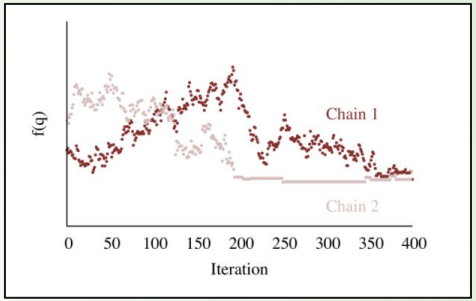
- The length of the warm-up stage:
  - Longer warm-up allows the chain to explore the posterior geometry more fully
  - Too short may lead to a mass matrix M and step size ε which is tuned too specifically to one region of the posterior, and does not generalise well to other regions

- The length of the sampling-stage:
  - Determines the maximum effective number of samples
  - You can run for longer:
    - if all diagnostics are good
    - if you require better precision as quantified by MCMC-SE

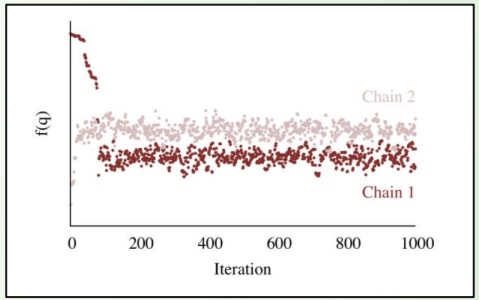
MCMC diagnostics

- Gelman Rubin statistic AKA *Rhat*
  - Has sampling converged onto a stable distribution?
  - If several chains are run (recommended!)
    - measures how well mixed different chains are
  - If one chain is run:
    - measures if first half of the chain well-mixed with the second
  - Recommended: Rhat < 1.05 for all parameters

Poorly mixed chains with Rhat >> 1



Chains not well-mixed with themselves or each-other



Chains are well-mixed internally, but not mixed or each-other

MCMC diagnostics

- Gelman Rubin statistic AKA *Rhat*
  - Has sampling converged onto a stable distribution?
  - If several chains are run (recommended!)
    - measures how well mixed different chains are
  - If one chain is run:
    - measures if first half of the chain well-mixed with the second
  - Recommended: Rhat < 1.05 for all parameters

- Effective sample size (ESS)
  - Estimate of the number of *independent* samples i.e. accounting for any autocorrelation in the chain
  - Recommended: ESS > 100 for all parameters

MCMC diagnostics

- Gelman Rubin statistic AKA *Rhat*
  - Has sampling converged onto a stable distribution?
  - If several chains are run (recommended!)
    - measures how well mixed different chains are
  - If one chain is run:
    - measures if first half of the chain well-mixed with the second
  - Recommended: Rhat < 1.05 for all parameters

- Effective sample size (ESS)
  - Estimate of the number of *independent* samples i.e. accounting for any autocorrelation in the chain
  - Recommended: ESS > 100 for all parameters

- Divergences:
  - N = 0