Lecture 2: Monte Carlo techniques

Data Science and Information Theory, ED127 course (2025)

Florent Leclercq

www.florent-leclercq.eu

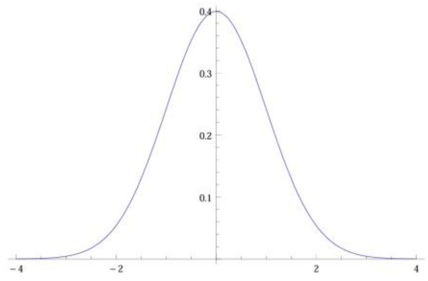
Institut d'Astrophysique de Paris
CNRS & Sorbonne Université

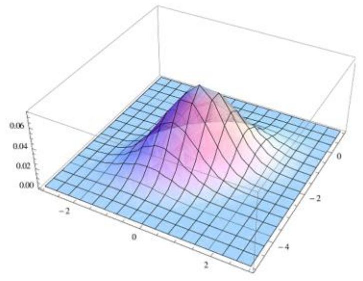
1 April 2025

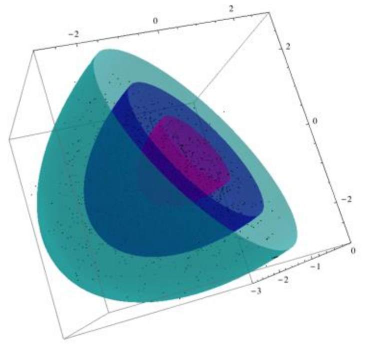
02 MONTE CARLO TECHNIQUES

Exploration of the posterior – Reporting inferences

- The output of a Bayesian analysis is a pdf: the posterior.
- The posterior cannot always be easily represented. Communication can take various forms:
  - Direct visualisation if the parameter space has sufficiently large dimension,

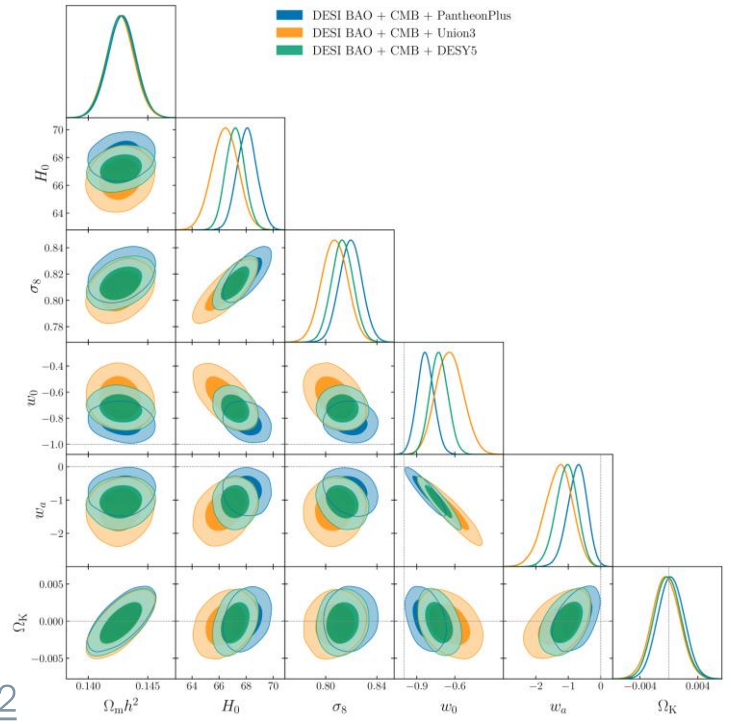






  - Credible regions (e.g. shortest interval containing 68% of the posterior probability),
    Warning: this has not the same meaning as a frequentist confidence interval (although the 2 might be formally identical)

- Statistical summaries of the posterior, e.g.
  - the mean, the median, or the mode of the distribution of each parameter, marginalising over all others;
  - its standard deviation;
  - the means and covariance matrices of some groups of parameters,
- "Corner plots"

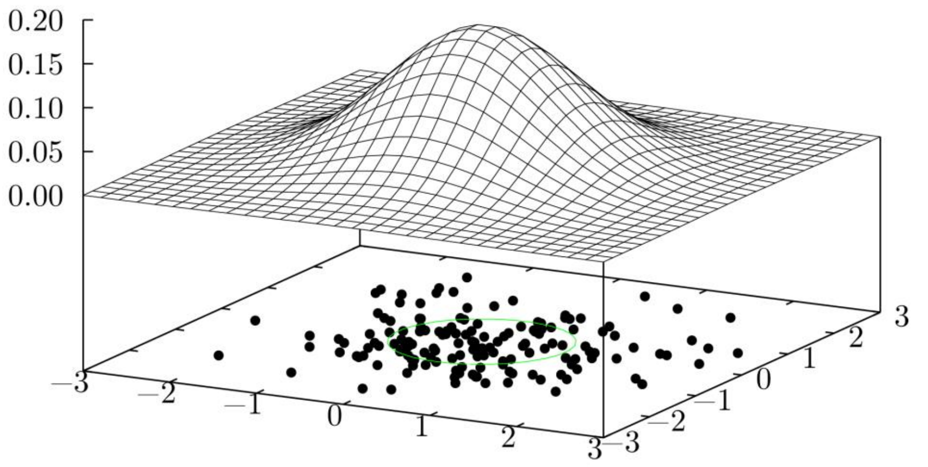


Leclercq (2015), chap. 3

DESI 2024 VI, 2404.03002

Exploration of the posterior: challenges

- The number of grid points increases exponentially with D: direct mapping of the posterior density is practically impossible for $D \geq 5$.
- Computing statistical summaries by marginalisation means integrating out other parameters:
  - Analytical integration is rarely possible (except for GRFs)
  - Even numerical integration is basically hopeless for $D > 5$.
- In high dimension, direct evaluation of the posterior is impossible and one has to rely on a numerical approximation: representing the posterior distribution by a set of samples.



$$p(\theta|d) \approx p_N(\theta|d) = \frac{1}{N}\sum_{i=1}^{N} \delta_{\mathrm{D}}(\theta - \theta_i) \quad \text{with} \quad \theta_i \curvearrowleft p(\theta|d)$$

Leclercq (2015), chap. 3

Living in a world made of samples

- Each sample is one “possible version of the truth”.
- The variation among different samples quantifies the uncertainty.
- In a world made of samples:
  - Multiplication is hard…

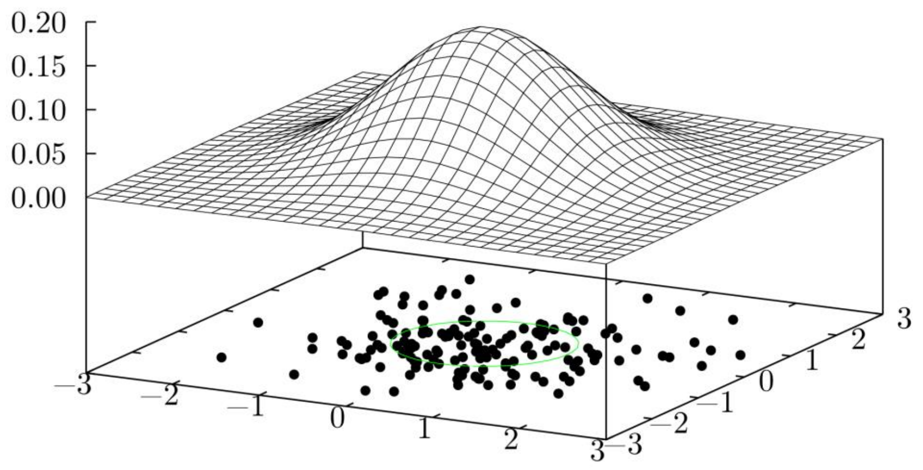
$$\left[\sum_{i=1}^{N_\mathrm{A}} \delta_\mathrm{D}(\theta-\theta_i)\right] \times \left[\sum_{j=1}^{N_\mathrm{B}} \delta_\mathrm{D}(\theta-\theta_j)\right] \quad \text{gives (almost certainly) zero!}$$

  - But integration is easy!

$$\langle\theta\rangle_{p(\theta|d)} = \int \theta\, p(\theta|d)\, \mathrm{d}\theta \approx \frac{1}{N}\sum_{i=1}^{N}\theta_i \quad \text{with} \quad \theta_i \curvearrowleft p(\theta|d)$$

More generally:

$$\langle f(\theta)\rangle_{p(\theta|d)} = \int f(\theta)\, p(\theta|d)\, \mathrm{d}\theta \approx \frac{1}{N}\sum_{i=1}^{N} f(\theta_i) \quad \text{with} \quad \theta_i \curvearrowleft p(\theta|d)$$



- In particular, marginalisation is trivial:

$$\langle\theta\rangle_{p(\theta,\varphi|d)} = \iint \theta\, p(\theta,\varphi|d)\, \mathrm{d}\theta\mathrm{d}\varphi = \int \theta\, p(\theta|d)\, \mathrm{d}\theta$$
$$\approx \frac{1}{N}\sum_{i=1}^{N}\theta_i \quad \text{with} \quad (\theta_i,\varphi_i) \curvearrowleft p(\theta,\varphi|d)$$

Marginalisation is achieved simply by “not looking” at the values of φ_i in the samples (θ_i,φ_i)

- This procedure works in arbitrarily high dimension.

Rao-Blackwell estimators

- Frequentist version: if $g(d)$ is any kind of estimator of a parameter θ, then the conditional expectation of $g(d)$ given $T(d)$, where $T(d)$ is a sufficient statistic, is typically a "better" estimator of θ, and is never "worse".
- More precisely, define:
  - $\delta(d)$ an "original estimator" of θ
  - $\delta_1(X) = E[\delta(d) \mid T(d)]$ the Rao-Blackwell "improved estimator" (it shall be observable, i.e. not depend on θ)
- Then: the mean squared error of the Rao-Blackwell estimator does not exceed that of the original estimator, i.e. $E[(\delta_1(X) - \theta)^2] \leq E[(\delta(X) - \theta)^2]$.
- Demonstration: the mean square error of the Rao-Blackwell estimator has the following decomposition:

$$E[(\delta_1(X) - \theta)^2] = E[(\delta(X) - \theta)^2] + E[\mathrm{Var}(\delta(d) \mid T(d))],$$

$$\text{and } E[\mathrm{Var}(\delta(d) \mid T(d))] \geq 0.$$

- Improving an estimator based on this procedure is called "Rao-Blackwellisation".

Rao-Blackwell estimators

- Bayesian version: suppose we have data d, an underlying signal s, and a property x which does not depend on the data when the signal is known, i.e. $p(x|s,d) = p(x|s)$ (s is a sufficient summary statistic of d).
- Further, suppose we have a way to generate samples of the joint posterior, $(x_i, s_i) \curvearrowleft p(x,s|d)$.
- The naïve estimator of the marginal pdf $p(x|d)$ is $p(x|d) \approx \frac{1}{N}\sum_{i=1}^{N} x_i \equiv \hat{x}$
- But we also have

$$p(x|d) = \int p(x,s|d)\,\mathrm{d}s = \int p(x|s,d)p(s|d)\,\mathrm{d}s = \int p(x|s)p(s|d)\,\mathrm{d}s$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} p(x|s_i) \equiv \hat{x}_1 \quad \text{with} \quad s_i \curvearrowleft p(s|d)$$: the Rao-Blackwell estimator of $p(x|d)$

- Then one can show that $\hat{x}_1$ is a “better” estimator of $p(x|d)$ than $\hat{x}$, in any reasonable sense.

Robert & Roberts, 2101.01011

Rao-Blackwell estimators

- In particular if $p(x|s_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left[-\frac{1}{2}\frac{(x-\mu_i)^2}{\sigma_i^2}\right]$
- The original estimator $p(x|d) \approx \frac{1}{N}\sum_{i=1}^{N} x_i \equiv \hat{x}$ is the sum of N independent Gaussian variables. It is a Gaussian with

  mean: $E(\hat{x}) = \frac{1}{N}\sum_{i=1}^{N}\mu_i$ variance: $E\left[(\hat{x}-E(\hat{x}))^2\right] = \frac{1}{N}\sum_{i=1}^{N}\sigma_i^2$

- The Rao-Blackwell estimator is $p(x|d) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\sqrt{2\pi\sigma_i}}\exp\left[-\frac{1}{2}\frac{(x-\mu_i)^2}{\sigma_i^2}\right] \equiv \hat{x}_1$. It is non-Gaussian but has:

mean: $E(\hat{x}_1) = \frac{1}{N}\sum_{i=1}^{N}\mu_i = E(\hat{x})$

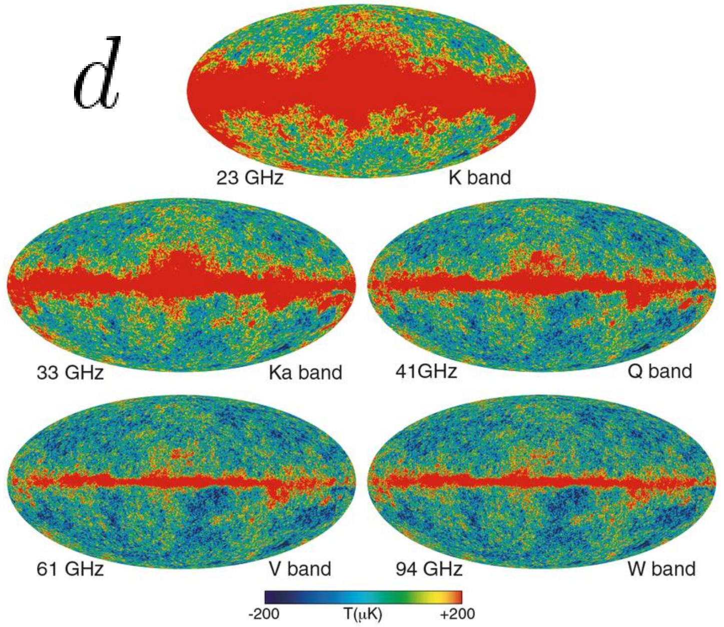
variance: $E\left[(\hat{x}_1 - E(\hat{x}_1))^2\right] \approx \frac{1}{N}\sum_{i=1}^{N}(\mu_i^2+\sigma_i^2) - \left(\frac{1}{N}\sum_{i=1}^{N}\mu_i\right)^2$

$$\leq \frac{1}{N}\sum_{i=1}^{N}\sigma_i^2 = E\left[(\hat{x}-E(\hat{x}))^2\right]$$

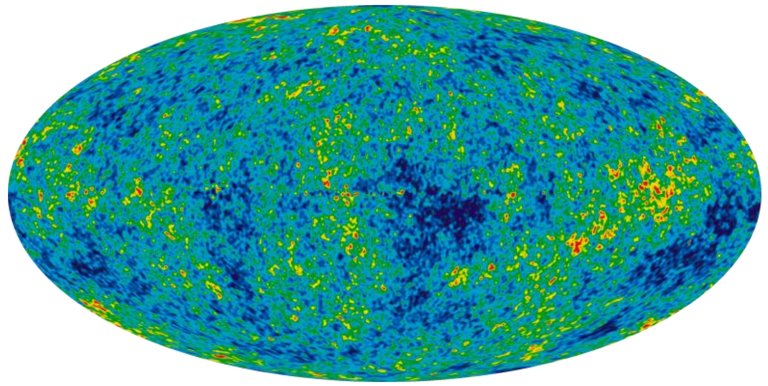
Rao-Blackwell estimators: example

- For analysis of the cosmic microwave background, we want to know the distribution of the power spectrum coefficients C_ℓ given the data d (frequency maps). The signal s is the cosmic microwave background map.

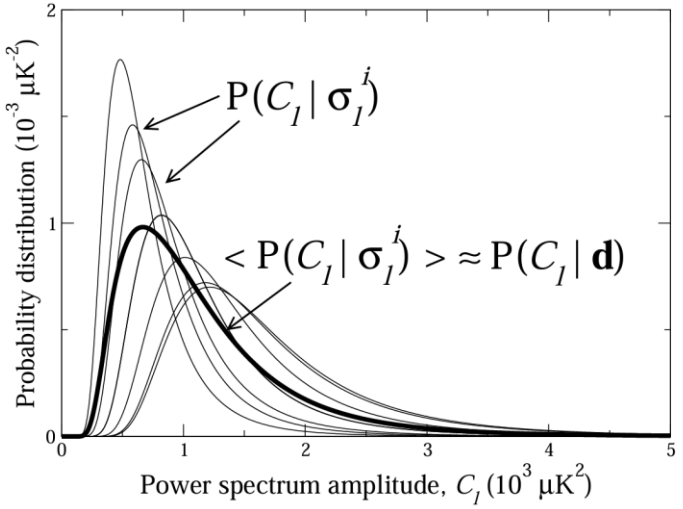
d



s



C_ℓ



$$p(C_\ell|s) = p(C_\ell|\sigma_\ell)$$

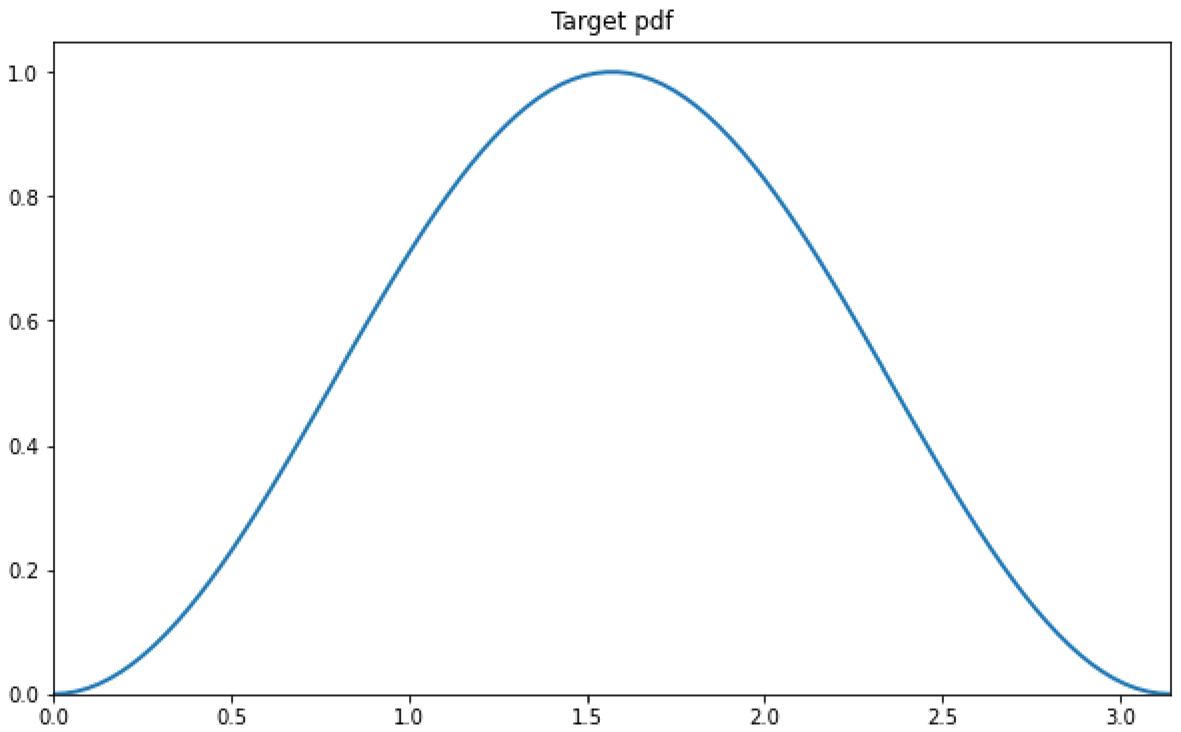
$$p(C_\ell|d) = \int p(C_\ell, s|d)\,\mathrm{d}s = \int p(C_\ell|s)p(s|d)\,\mathrm{d}s \overset{\downarrow}{=} \int p(C_\ell|\sigma_\ell)p(\sigma_\ell|d)\,\mathrm{d}\sigma_\ell \approx \frac{1}{N}\sum_{i=1}^{N} p(C_\ell|\sigma_\ell^i)$$

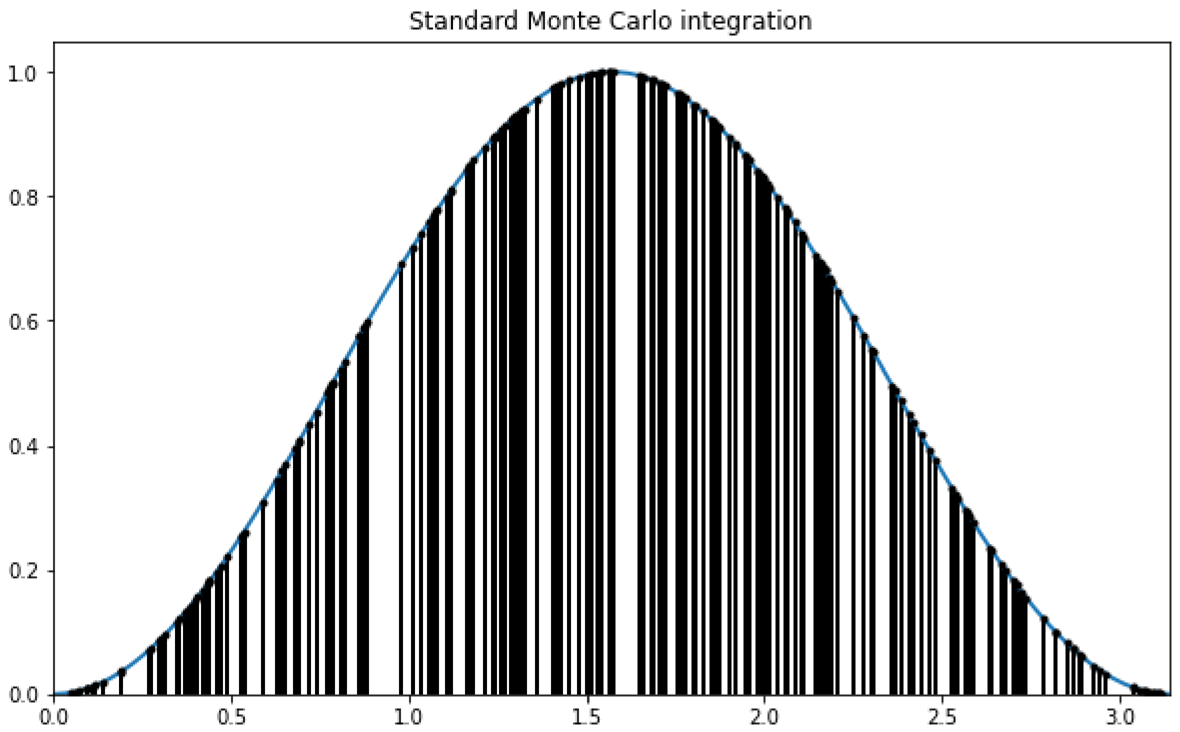
Wandelt, Larson & Lakshminarayanan (2003), astro-ph/0310080

MONTE CARLO INTEGRATION

Monte Carlo integration: standard sampling

Exercise: Monte Carlo Integration





$$I = \int_a^b f(x)\,\mathrm{d}x$$

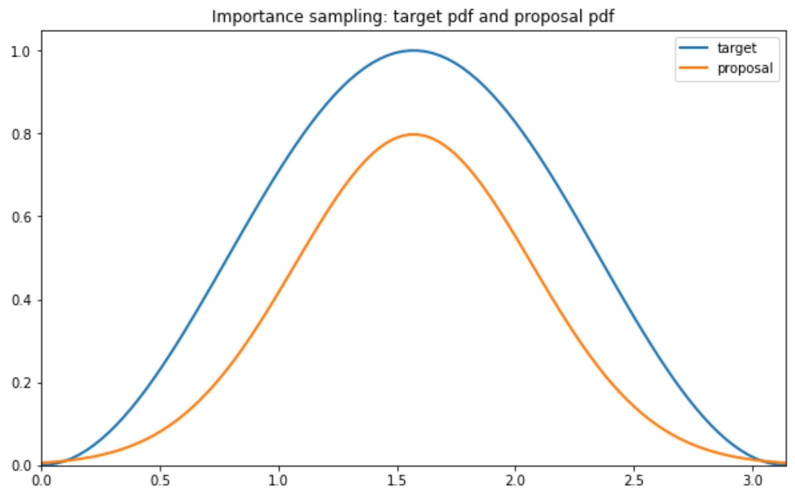
$$I \approx \frac{b-a}{N_{\mathrm{samples}}} \sum_{i=1}^{N_{\mathrm{samples}}} x_i$$

```
In [6]:  trueI=quad(target_pdf,a,b)[0]
         trueI

Out[6]:  1.5707963267948966
```

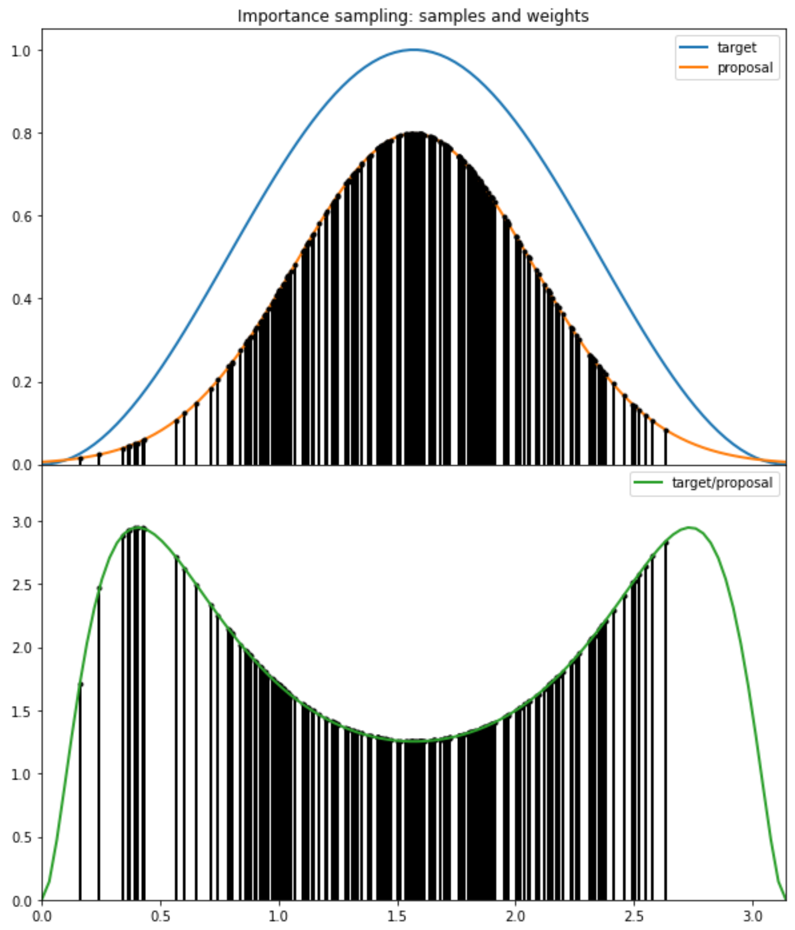
```
In [9]:  StandardMonteCarloI=np.sum(samples)*(b-a)/Nsamp
         StandardMonteCarloI

Out[9]:  1.660402945484072
```

Monte Carlo integration: importance sampling

- We draw samples from a proposal pdf, designed to be as close as possible to the target pdf. We then assign each sample a weight proportional to its likelihood divided by its prior probability.



$$I \approx \frac{\sum_{i=1}^{N_{\text{samples}}} x_i w_i}{\sum_{i=1}^{N_{\text{samples}}} w_i}$$



```
In [14]:  ImportanceI=np.average(samples,weights=weights)
          ImportanceI

Out[14]:  1.5100957559182684
```

Monte Carlo integration: importance resampling

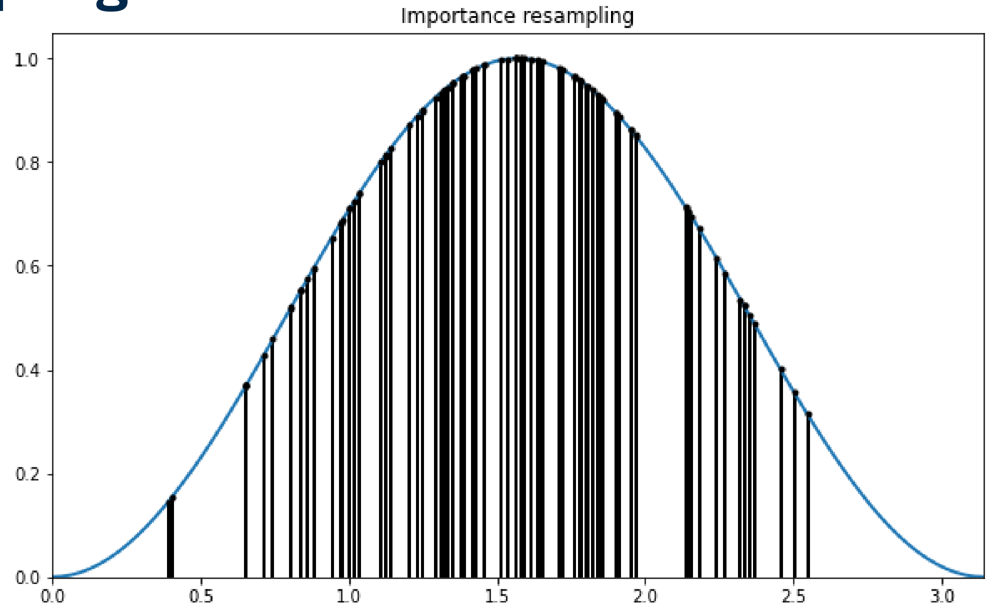
- A problem with importance sampling is the situation in which all but one of the weights are close to zero. To avoid with situation, we can do importance resampling. We draw N_{resamp} new samples from the current sample set with probabilities proportional to their weights. We replace the current samples with this new set, and the the current weights by $1/N_{\text{resamp}}$ (drawing according to the importance weight replaces likelihoods by frequencies).

```
In [15]: Nresamp=100
         normalizedweights=weights/np.sum(weights)
         resamples=np.random.choice(samples, size=Nresamp, replace=True, p=normalizedweights)
         reweights=1./Nresamp*np.ones(Nresamp)
```

- Weights are then updated given their likelihood, as in the previous importance sampling step.

```
In [16]: reweights*=target_pdf(resamples)/(1./Nresamp)
```

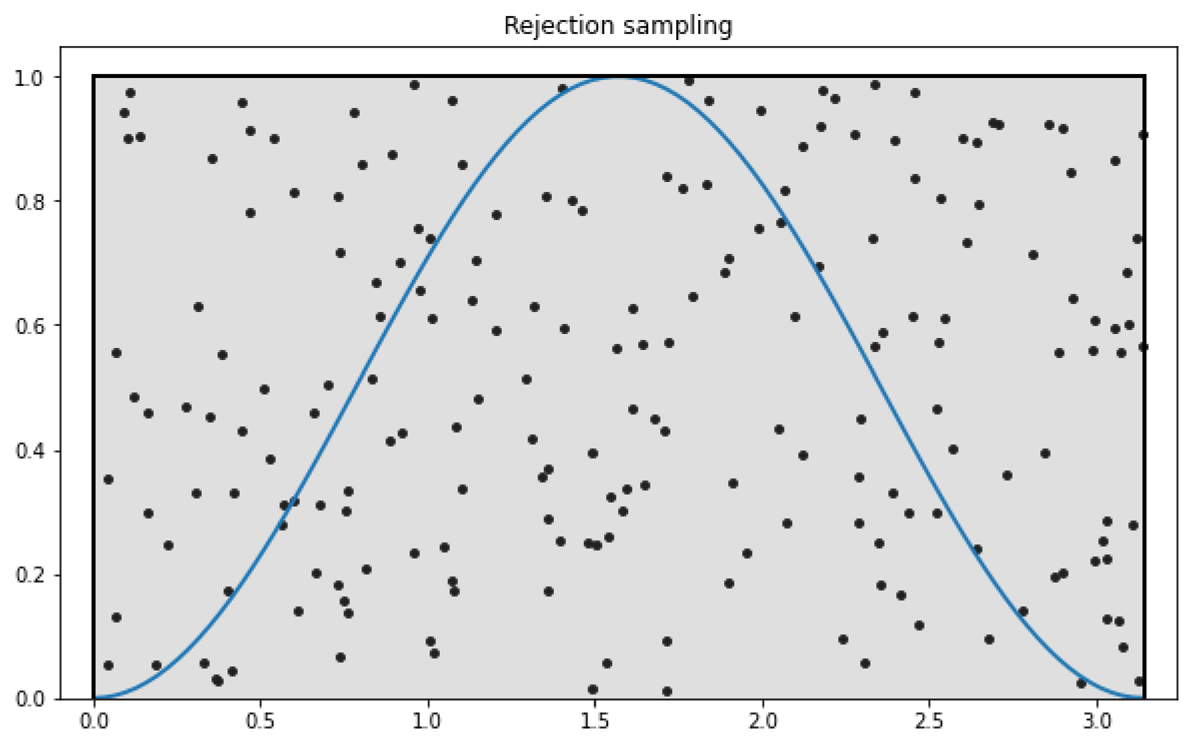
$$I \approx \frac{\sum_{i=1}^{N_{\text{samples}}} x_i w_i}{\sum_{i=1}^{N_{\text{samples}}} w_i}$$

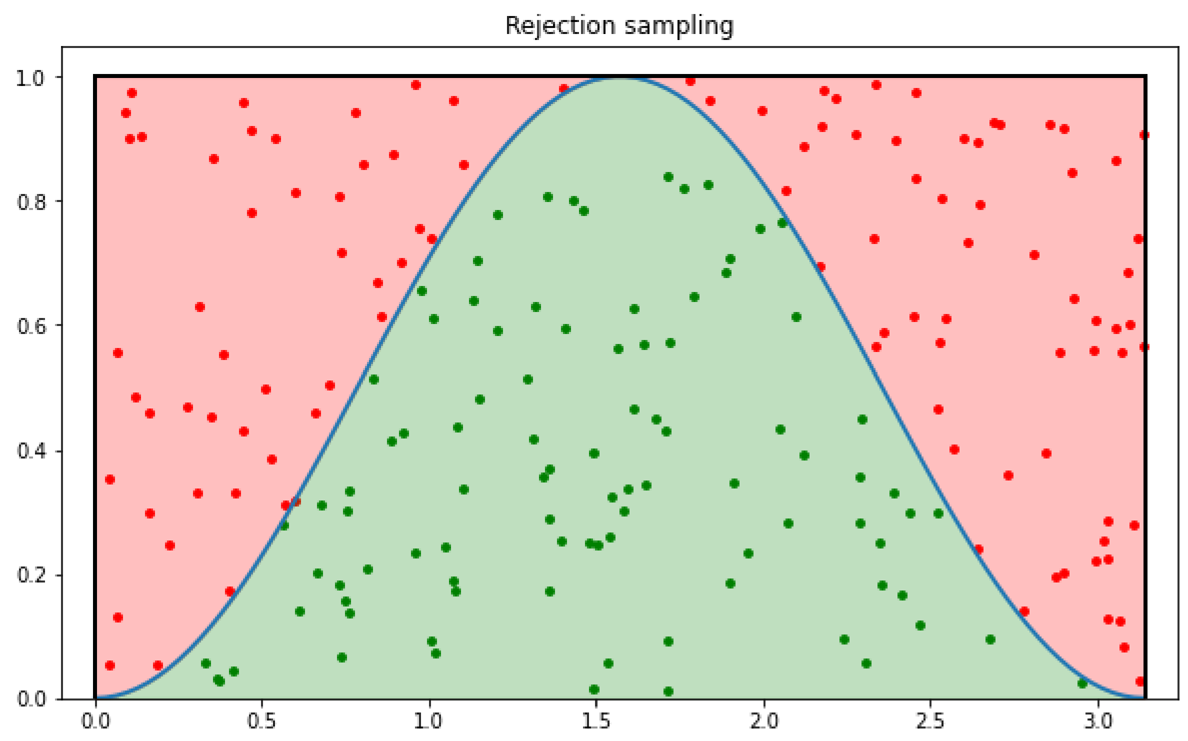


```
In [18]: ImportanceReI=np.average(resamples,weights=reweights)
         ImportanceReI

Out[18]: 1.531675529534044
```

- Iterating this procedure yields the Sequential Importance Resampling (SIR) algorithm, which is a simple "particle filter" or "Sequential Monte Carlo" algorithm.

Monte Carlo integration: rejection sampling





```
In [23]:  fraction=float(len(accepted_samples))/Nsamp
          fraction

Out[23]:  0.465

In [24]:  fraction=float(len(accepted_samples))/Nsamp
          RejectionI=fraction*upperbound*(b-a)
          RejectionI

Out[24]:  1.460840583919254
```

MARKOV CHAIN MONTE CARLO

Markov Chain Monte Carlo (MCMC): the Metropolis-Hastings algorithm

- The Markov property:
  - The useful information for predicting the future is entirely contained in the present state of the process and does not depend on past states (the system has no "memory").
  - Mathematically: The conditional probability distribution of future states, given past states and the present state, depends only on the present state and not on past states:

$$p(\boldsymbol{x}_{n+1} | \{\boldsymbol{x}_i\}_{1\leq i\leq n}) = p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_n)$$

- Metropolis-Hastings algorithm:

```
begin
    initialise $\boldsymbol{x}_{(0)}$;
    for $i = 1$ to $n$ do
        $\boldsymbol{x}^* \curvearrowleft q(\boldsymbol{x}^*|\boldsymbol{x})$ (proposal distribution);
        $\alpha \curvearrowleft \mathcal{U}(0,1)$ (uniform distribution);
        if $\alpha < \min[1, r(\boldsymbol{x},\boldsymbol{x}^*)]$ then
            $\boldsymbol{x}_{(i)} = \boldsymbol{x}^*$;
        else
            $\boldsymbol{x}_{(i)} = \boldsymbol{x}_{(i-1)}$;
        end
    end
    return $(\boldsymbol{x}_{(0)}, \dots, \boldsymbol{x}_{(n)})$;
end
```

General case*: $r(\boldsymbol{x},\boldsymbol{x}^*) \equiv \dfrac{p(\boldsymbol{x}^*)}{p(\boldsymbol{x})}\dfrac{q(\boldsymbol{x}|\boldsymbol{x}^*)}{q(\boldsymbol{x}^*|\boldsymbol{x})}$ (Hastings ratio)

Particular case: $r(\boldsymbol{x},\boldsymbol{x}^*) = \dfrac{p(\boldsymbol{x}^*)}{p(\boldsymbol{x})}$ (Metropolis update)

for a symmetric proposal pdf: $q(\boldsymbol{x}^*|\boldsymbol{x}) = q(\boldsymbol{x}|\boldsymbol{x}^*)$

* This is only a very simplified and practical guide to MCMC. A more extensive treatment requires introducing the notions *stationarity* and *global/detailed balance*.

  - It is possible to prove that the chain has the target distribution as its stationary distribution, i.e. elements of the chain (asymptotically) become correlated samples of $p(\boldsymbol{x})$.

Metropolis-Hasting algorithm: implementation

- Metropolis-Hastings algorithm:

```
begin
    initialise $\boldsymbol{x}_{(0)}$;
    for $i = 1$ to $n$ do
        $\boldsymbol{x}^* \curvearrowleft q(\boldsymbol{x}^*|\boldsymbol{x})$ (proposal distribution);
        $\alpha \curvearrowleft \mathcal{U}(0,1)$ (uniform distribution);
        if $\alpha < \min\left[1, r(\boldsymbol{x}, \boldsymbol{x}^*)\right]$ then
            $\boldsymbol{x}_{(i)} = \boldsymbol{x}^*$;
        else
            $\boldsymbol{x}_{(i)} = \boldsymbol{x}_{(i-1)}$;
        end
    end
    return $(\boldsymbol{x}_{(0)}, \ldots, \boldsymbol{x}_{(n)})$;
end
```

```python
def MH_sampler(Ntries,theta_start,Ntrials,Nsuccesses,lh,prior,proposal_sigma):
    Naccepted=0
    samples=np.zeros(Ntries+1)
    samples[0]=theta_start
    theta=theta_start
    for i in range(Ntries):
        theta_p = theta + proposal_pdf(proposal_sigma).rvs()
        # the Gaussian proposal pdf satisfies the detailed balance equation, so the
        # acceptance ratio simplifies to the Metropolis ratio
        a = min(1, target_pdf(theta_p,Ntrials,Nsuccesses,lh,prior)/target_pdf(theta,Ntrials,Nsuccesses,lh,prior))
        u = np.random.uniform()
        if u < a:
            Naccepted+=1
            theta=theta_p
        samples[i+1] = theta
    return Naccepted, samples
```

General case: $r(\boldsymbol{x}, \boldsymbol{x}^*) \equiv \dfrac{p(\boldsymbol{x}^*)}{p(\boldsymbol{x})} \dfrac{q(\boldsymbol{x}|\boldsymbol{x}^*)}{q(\boldsymbol{x}^*|\boldsymbol{x})}$ (Hastings ratio)

Particular case: $r(\boldsymbol{x}, \boldsymbol{x}^*) = \dfrac{p(\boldsymbol{x}^*)}{p(\boldsymbol{x})}$ (Metropolis update)

for a symmetric proposal pdf: $q(\boldsymbol{x}^*|\boldsymbol{x}) = q(\boldsymbol{x}|\boldsymbol{x}^*)$

A toy Bayesian problem

Exercise: Metropolis-Hastings

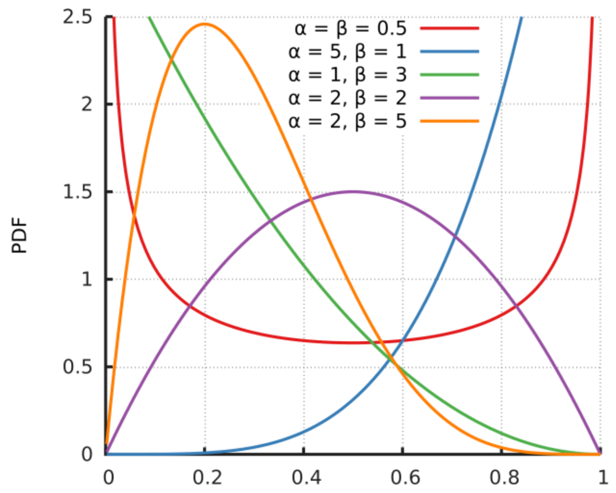
- Considered problem: a Bernoulli experiment (N_{trials} independent trials each with a probability of success θ)
- The likelihood for this problem is a binomial distribution:

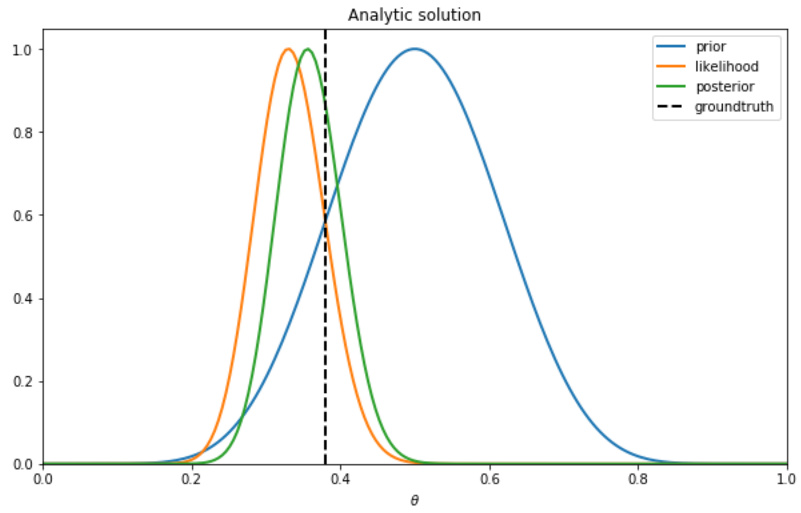
$$p(N_{\text{successes}}, N_{\text{trials}}, \theta) = \binom{N_{\text{trials}}}{N_{\text{successes}}} \theta^{N_{\text{successes}}} (1-\theta)^{N_{\text{trials}} - N_{\text{successes}}}$$

- Analytic result: the beta distribution gives a family of conjugate priors, meaning that if the prior is $\mathcal{B}(\alpha, \beta)$, then the posterior is $\mathcal{B}(\alpha', \beta')$ with $\alpha' = \alpha + N_{\text{sucesses}}$
$\beta' = \beta + N_{\text{trials}} - N_{\text{sucesses}}$

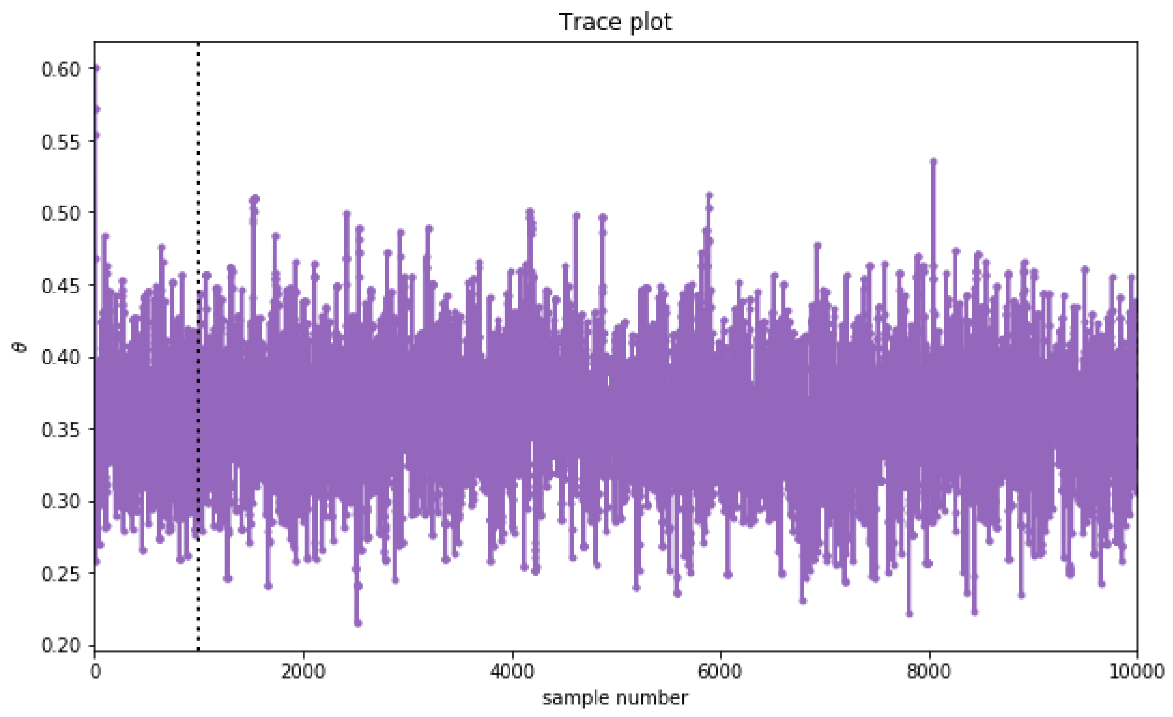
$$\mathcal{B}(\alpha, \beta)(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha, \beta)}$$

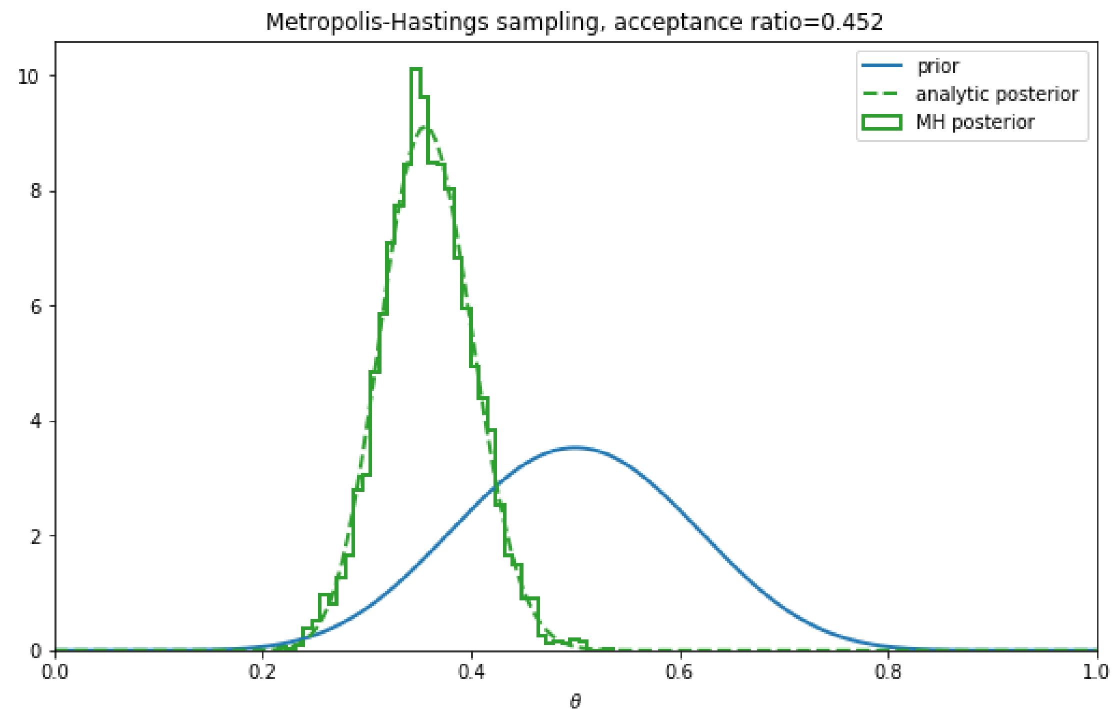
with $\mathrm{B}(\alpha, \beta) \equiv \dfrac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$





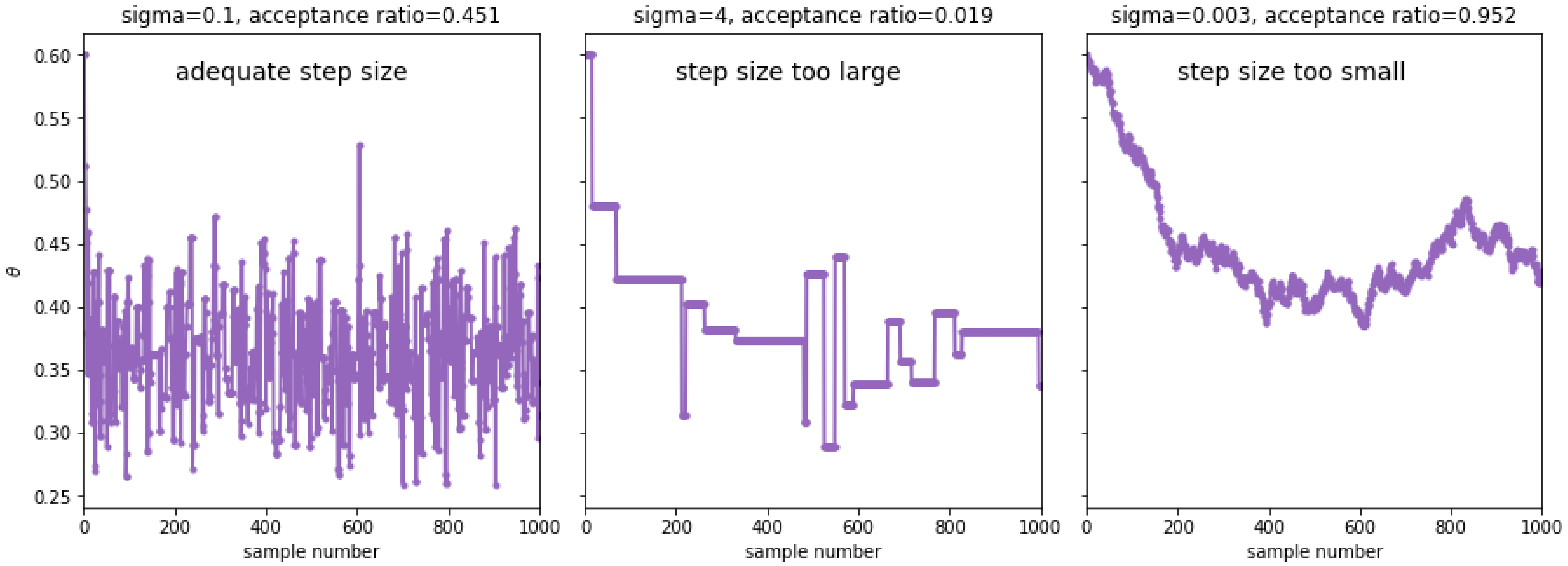
Diagnostics of Markov chains: burn-in





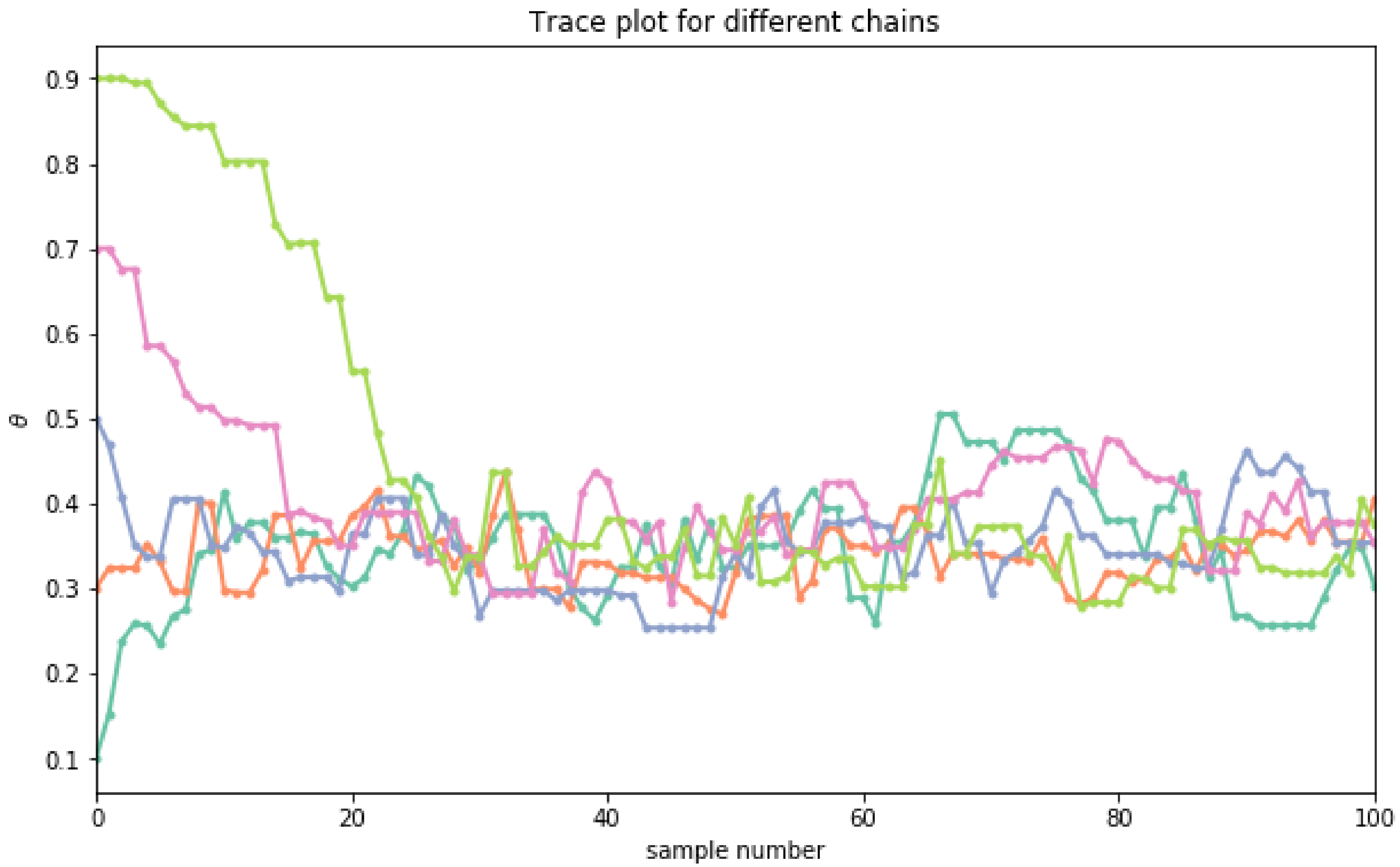
Diagnostics of Markov chains: trace plots

- Adjusting the proposal distribution (here by changing the step size):



Diagnostics of Markov chains: mixing

- Several independent chains, different starting points:



Diagnostics of Markov chains: convergence – the Gelman-Rubin test

- Parameters:
  - m : number of chains
  - n: length of chains

$$\bar{\psi}_{.j} = \frac{1}{n}\sum_{i=1}^{n}\psi_{ij}$$

- Definitions:
  - “between”-chains variance: $B \equiv \frac{n}{m-1}\sum_{j=1}^{m}\left(\bar{\psi}_{.j} - \bar{\psi}_{..}\right)^2$

$$\bar{\psi}_{..} = \frac{1}{m}\sum_{j=1}^{m}\bar{\psi}_{.j}$$

  - “within”-chains variance: $W \equiv \frac{1}{m}\sum_{j=1}^{m} s_j^2$

$$s_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(\psi_{ij} - \bar{\psi}_{.j}\right)^2$$

- Estimators of the marginal posterior variance of the estimand (for each parameter):
  - $\widehat{\text{var}}^- \equiv W$: underestimates the variance
  - $\widehat{\text{var}}^+ \equiv \frac{n}{n-1}W + \frac{1}{n}B$: overestimates the variance
- Gelman-Rubin test:
  - Potential scale reduction factor: $\widehat{R} \equiv \sqrt{\frac{\widehat{\text{var}}^+}{\widehat{\text{var}}^-}}$
  - Test: $\widehat{R} \to 1$ as $n \to \infty$
  - Typically, one aims for $\widehat{R} - 1 \lesssim 10^{-2}$ for all parameters.

Gelman *et al.*, Bayesian Data Analysis (third edition), p. 284-285

MARKOV CHAIN MONTE CARLO BEYOND METROPOLIS-HASTINGS

Slice sampling

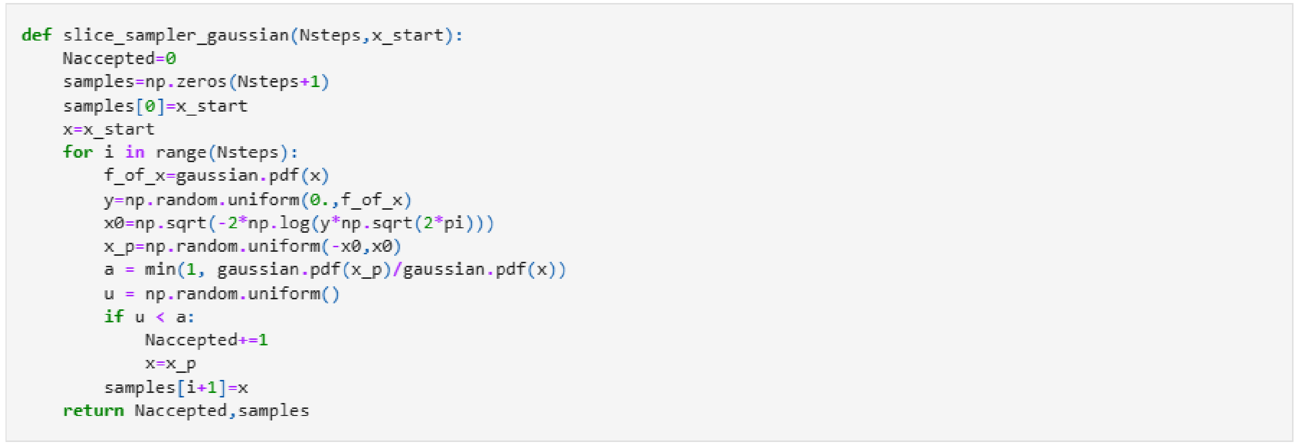
1. Drawing y

Starting from an initial x, we draw y uniformly in $[0, f(x)]$.

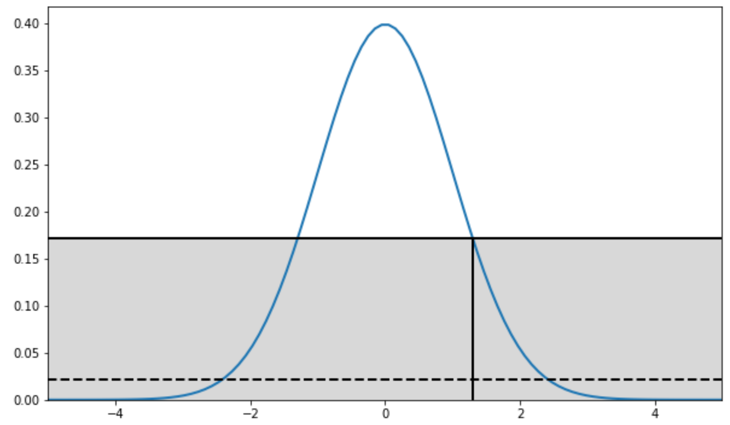
2. Drawing a new x

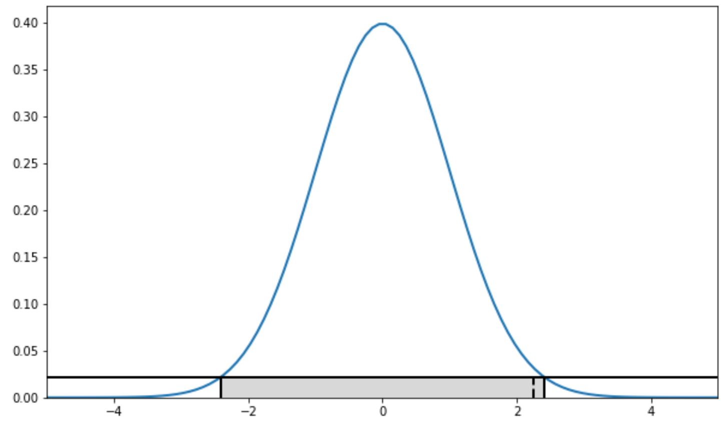
We draw x uniformly in the “slice” where $f(x) \geq y$. This proposed sample is accepted or rejected according to the usual Hastings rule.

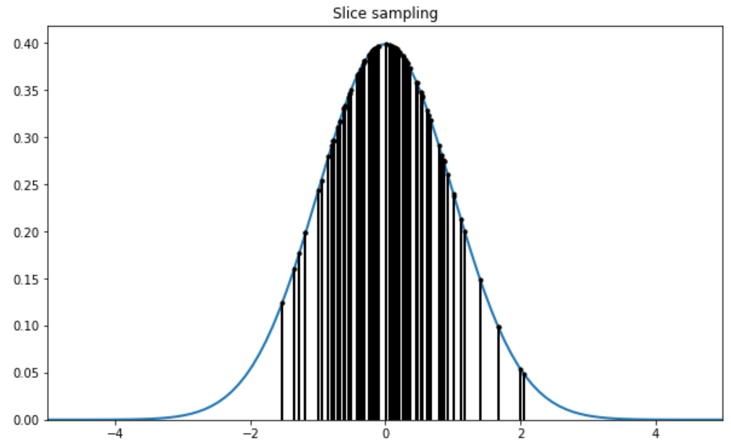
3. Iterate!



```
def slice_sampler_gaussian(Nsteps,x_start):
    Naccepted=0
    samples=np.zeros(Nsteps+1)
    samples[0]=x_start
    x=x_start
    for i in range(Nsteps):
        f_of_x=gaussian.pdf(x)
        y=np.random.uniform(0.,f_of_x)
        x0=np.sqrt(-2*np.log(y*np.sqrt(2*pi)))
        x_p=np.random.uniform(-x0,x0)
        a = min(1, gaussian.pdf(x_p)/gaussian.pdf(x))
        u = np.random.uniform()
        if u < a:
            Naccepted+=1
            x=x_p
        samples[i+1]=x
    return Naccepted,samples
```



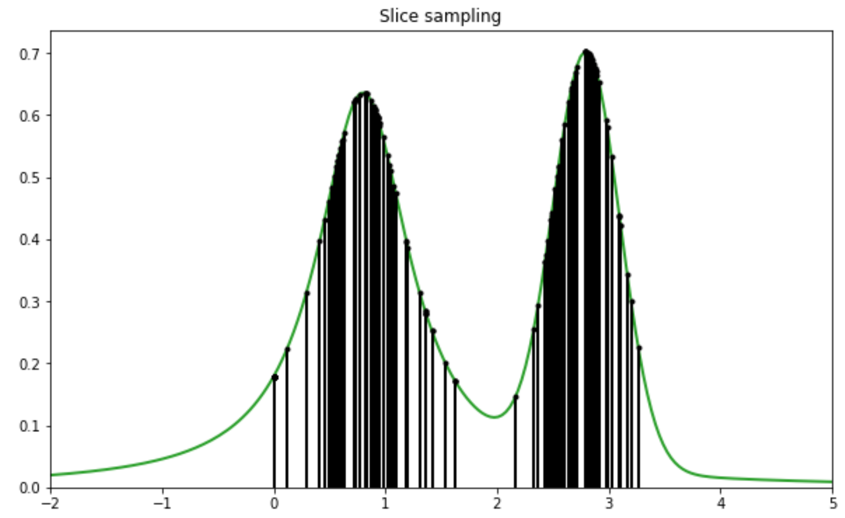




Slice sampling for multimodal pdfs

Exercise: Slice sampling

```python
def slice_sampler(target_pdf,Nsteps,x_start,x_width):
    Naccepted=0
    samples=np.zeros(Nsteps+1)
    samples[0]=x_start
    x=x_start
    for i in range(Nsteps):
        y=np.random.uniform(0, target_pdf(x))
        lb=x
        rb=x
        # we build the approximate slice by expanding around the current x
        while y<target_pdf(lb):
            lb-=x_width
        while y<target_pdf(rb):
            rb+=x_width
        # we draw a new x
        x_p=np.random.uniform(lb,rb)
        if target_pdf(x_p)>y:
            # x_p was in the slice, we keep it as a proposed sample
            # slice sampling satisfies detailed balance
            a = min(1, target_pdf(x_p)/target_pdf(x))
            u = np.random.uniform()
            if u < a:
                Naccepted+=1
                x=x_p
            samples[i+1]=x
        else:
            # x was not in the slice, we adjust the boundaries of the approximate slice
            if np.abs(x-lb)<np.abs(x-rb):
                lb = x
            else:
                rb = x
    return Naccepted,samples
```

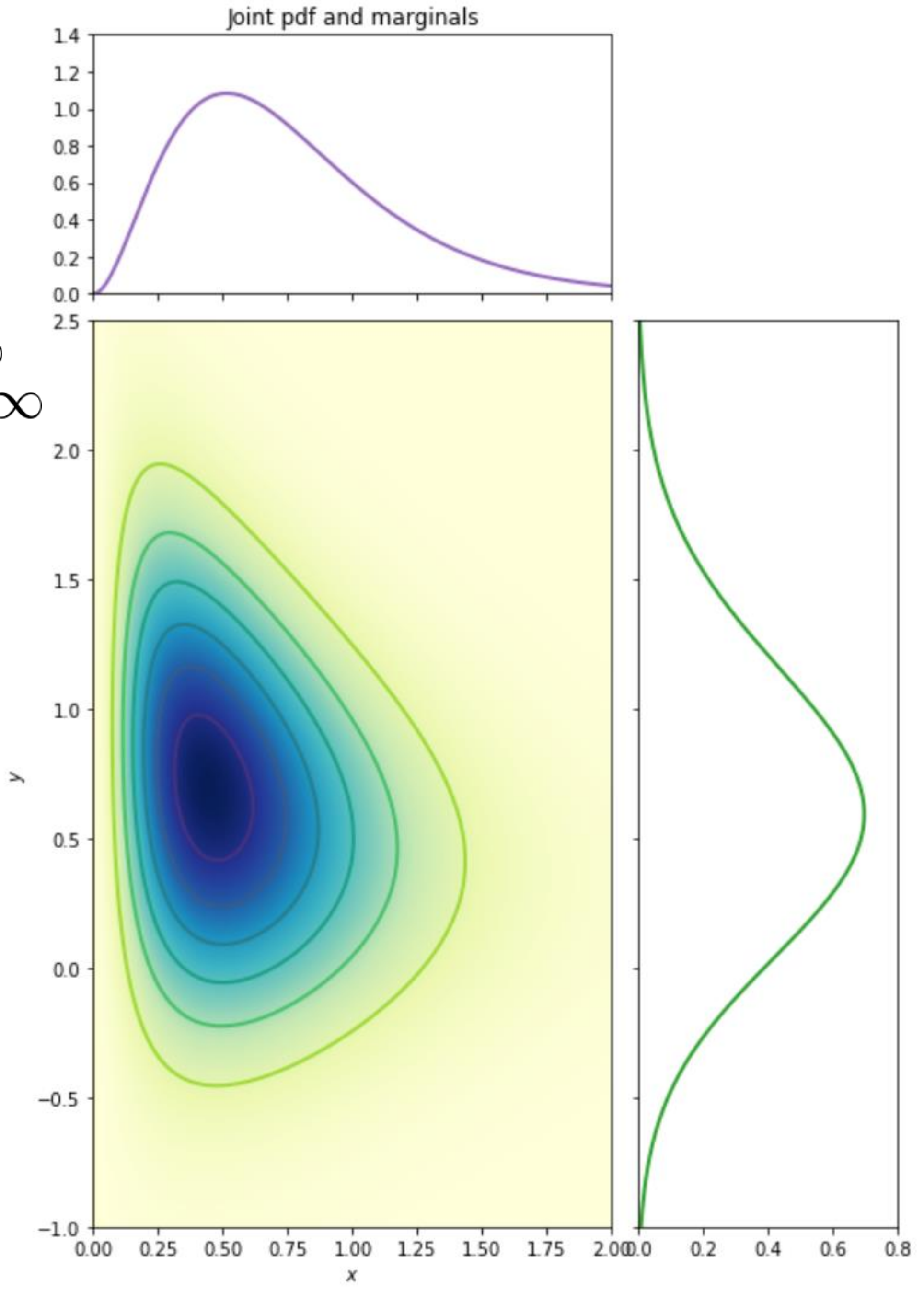


A two-dimensional test pdf

- Joint probability distribution:

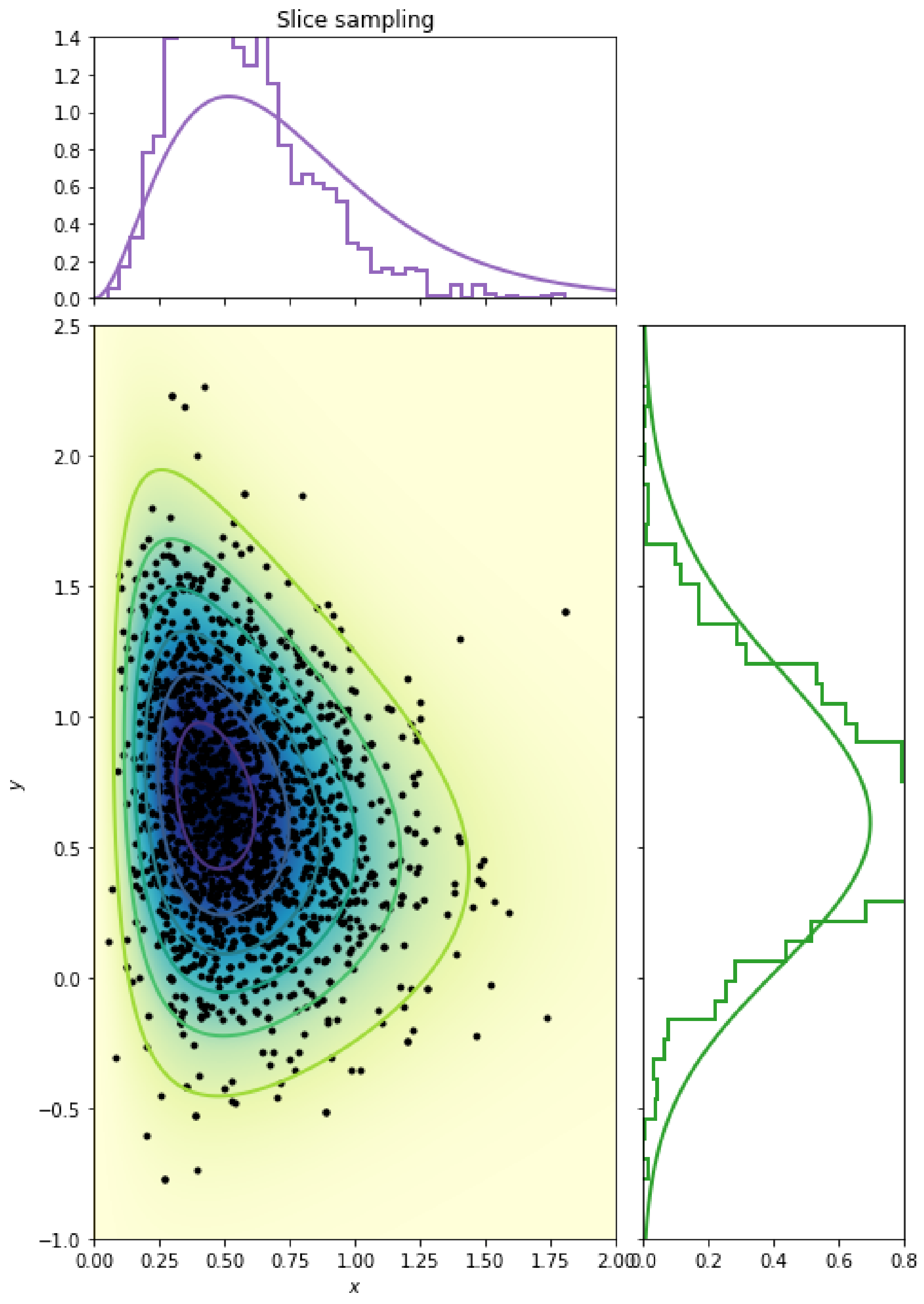
$$p(x, y) \propto x^2 \exp\left(-xy^2 - y^2 + 2y - 4x\right)$$

$$\begin{cases} 0 < x < +\infty \\ -\infty < y < +\infty \end{cases}$$



Slice sampler

```python
def slice_sampler_2D(target_pdf,Nsteps,x_start,xmin,xmax,y_start,ymin,ymax):
    Naccepted=0
    samples_x=np.zeros(Nsteps+1)
    samples_y=np.zeros(Nsteps+1)
    samples_x[0]=x_start
    samples_y[0]=y_start
    x=x_start
    y=y_start
    for i in range(Nsteps):
        z=np.random.uniform(0, target_pdf(x,y))

        # we draw a new (x,y) uniformly in the rectangle ([xmin,xmax],[ymin,ymax])
        # this may be inefficient. alternatively, one could adaptively define a 2D rectangle
        # (cf. hyperrectangle slice sampling)
        x_p=np.random.uniform(xmin,xmax)
        y_p=np.random.uniform(ymin,ymax)
        # we keep only points that are in the slice
        while target_pdf(x_p,y_p)<z:
            x_p=np.random.uniform(xmin,xmax)
            y_p=np.random.uniform(ymin,ymax)

        # (x_p,y_p) was in the slice, we keep it as a proposed sample
        # slice sampling satisfies detailed balance
        a = min(1, target_pdf(x_p,y_p)/target_pdf(x,y))
        u = np.random.uniform()
        if u < a:
            Naccepted+=1
            x=x_p
            y=y_p
        samples_x[i+1]=x
        samples_y[i+1]=y
    return Naccepted,samples_x,samples_y
```



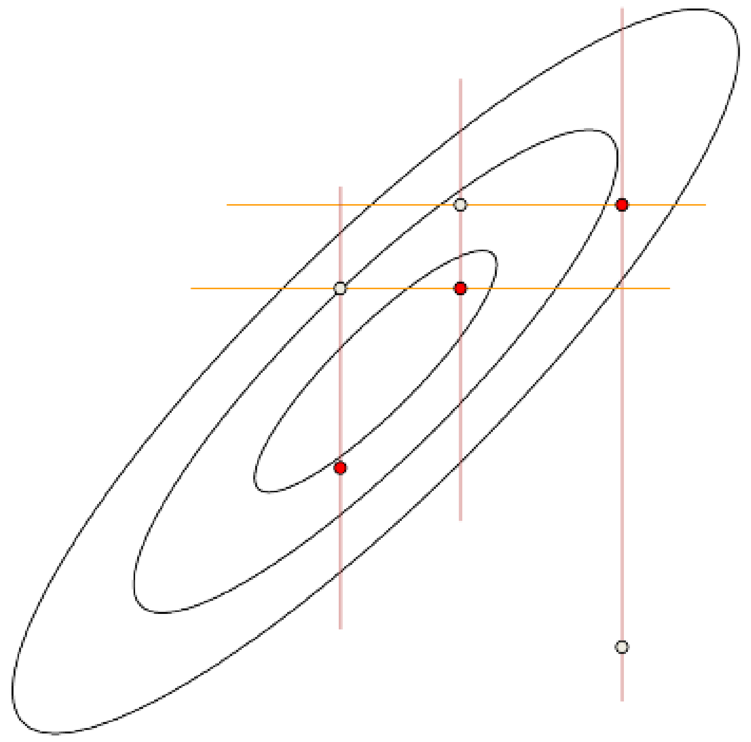
MCMC beyond Metropolis-Hastings

Exercise: Gibbs sampling

- Shortcomings of standard Metropolis-Hastings:
  - Tuning of proposal distributions
  - Curse of dimensionality

- Gibbs sampling:
  - Uses conditionals of the target pdf
  - Is a special case of Metropolis-Hastings with acceptance ratio unity:

$$r(x,y) = \frac{p(x^*,y)q(x,y)}{p(x,y)q(x^*,y)} = \frac{p(y)p(x^*|y)p(x|y)}{p(y)p(x|y)p(x^*|y)} = 1$$



A two-dimensional test pdf

- Joint probability distribution:

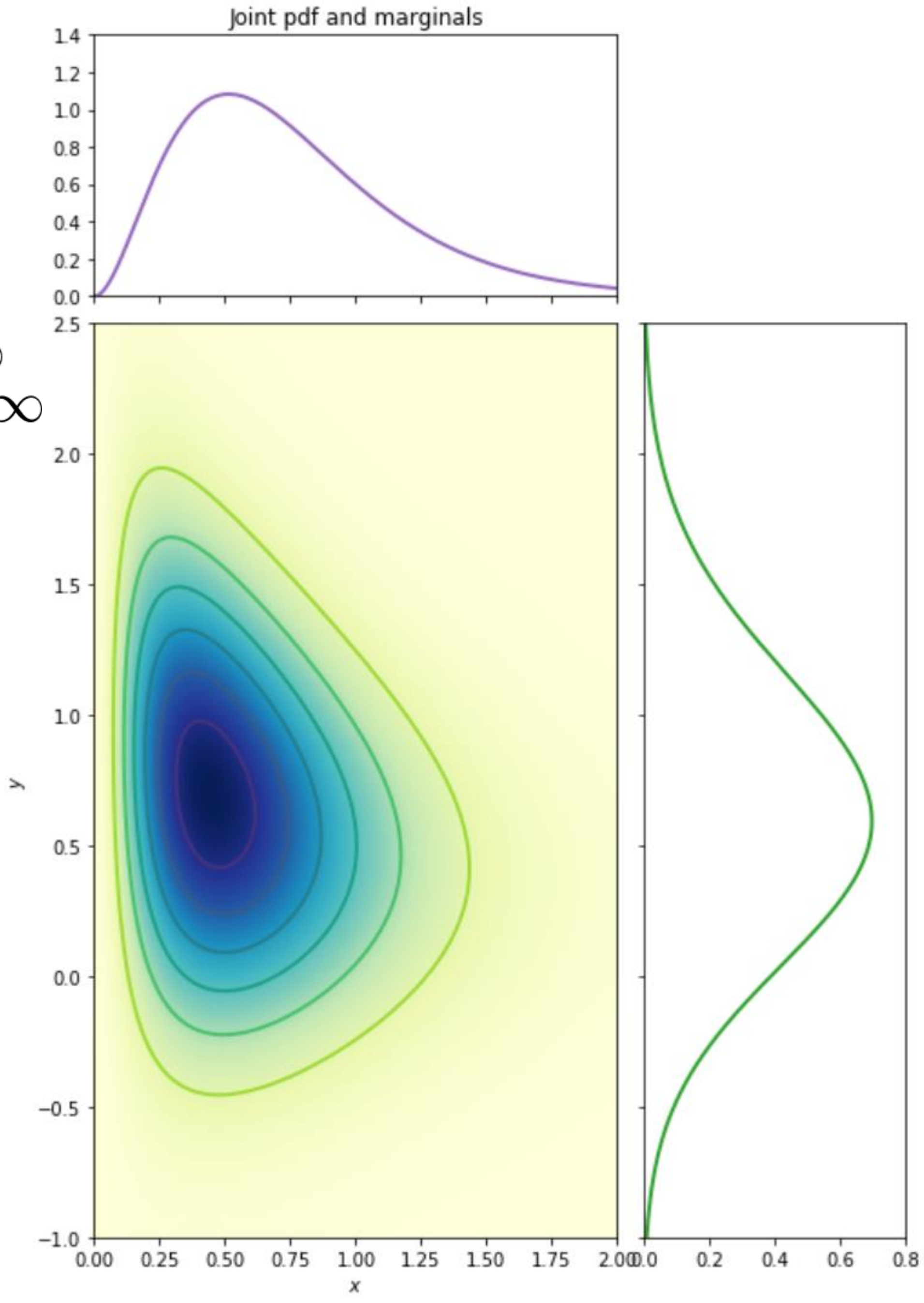
$$p(x,y) \propto x^2 \exp\left(-xy^2 - y^2 + 2y - 4x\right)$$

$$\begin{cases} 0 < x < +\infty \\ -\infty < y < +\infty \end{cases}$$

- Marginals:

$$p(x) \propto x^2 \mathrm{e}^{-4x} \frac{1}{\sqrt{x+1}} \mathrm{e}^{\frac{1}{x+1}}$$

$$p(y) \propto \frac{\mathrm{e}^{-y^2+2y}}{(y^2+4)^3}$$



A two-dimensional test pdf

- Joint probability distribution:

$$p(x,y) \propto x^2 \exp\left(-xy^2 - y^2 + 2y - 4x\right)$$

$$\begin{cases} 0 < x < +\infty \\ -\infty < y < +\infty \end{cases}$$

- Marginals:

$$p(x) \propto x^2 \mathrm{e}^{-4x} \frac{1}{\sqrt{x+1}} \mathrm{e}^{\frac{1}{x+1}}$$

$$p(y) \propto \frac{\mathrm{e}^{-y^2+2y}}{(y^2+4)^3}$$

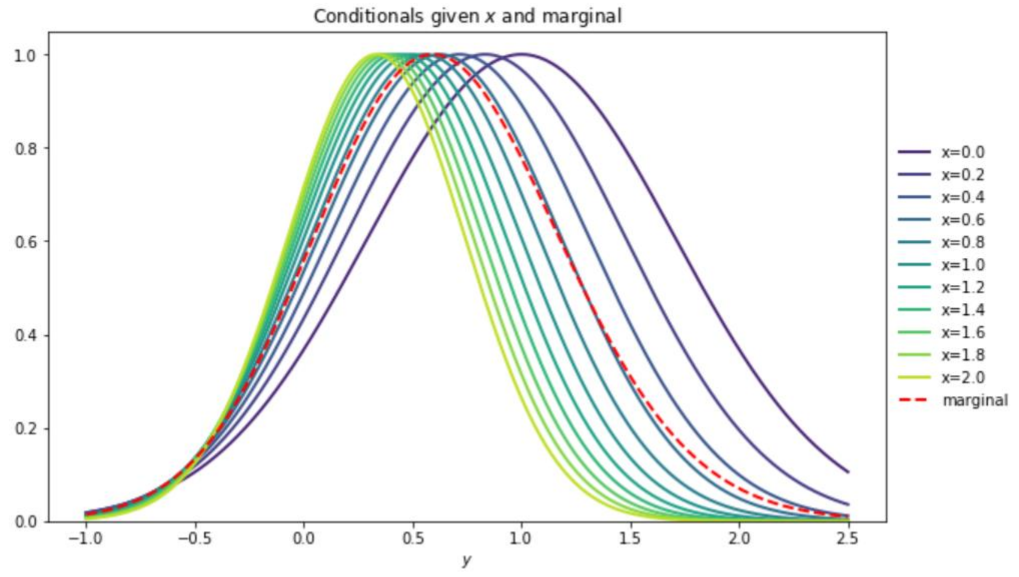
- Conditionals:

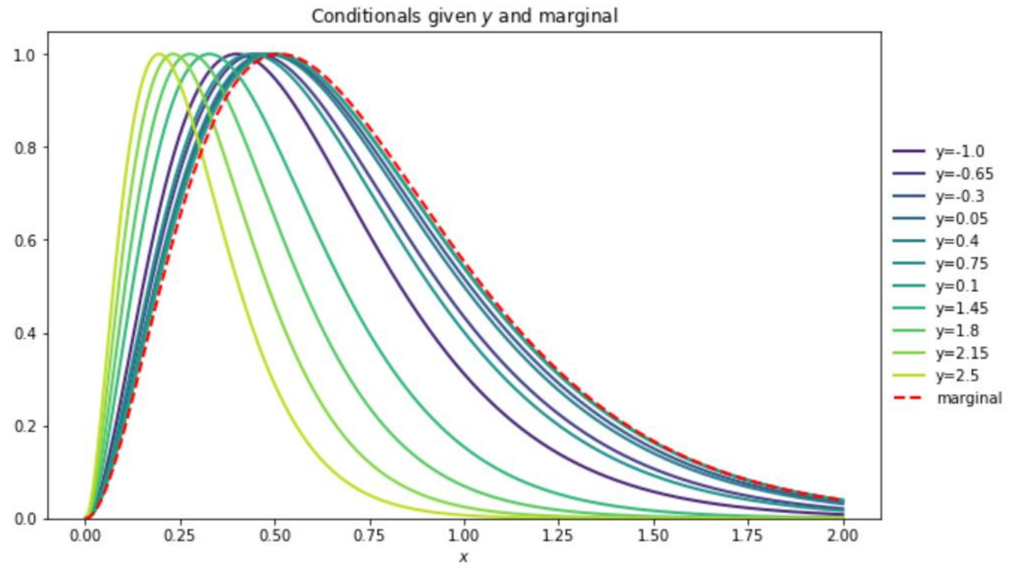
$$p(y|x) = \mathcal{G}\left(\mu = \frac{1}{1+x}, \sigma^2 = \frac{1}{2(1+x)}\right)(y)$$

with $\mathcal{G}\left(\mu, \sigma^2\right)(t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\frac{(t-\mu)^2}{\sigma^2}\right]$

$$p(x|y) = \Gamma(k = 3, \theta = y^2 + 4)(x)$$

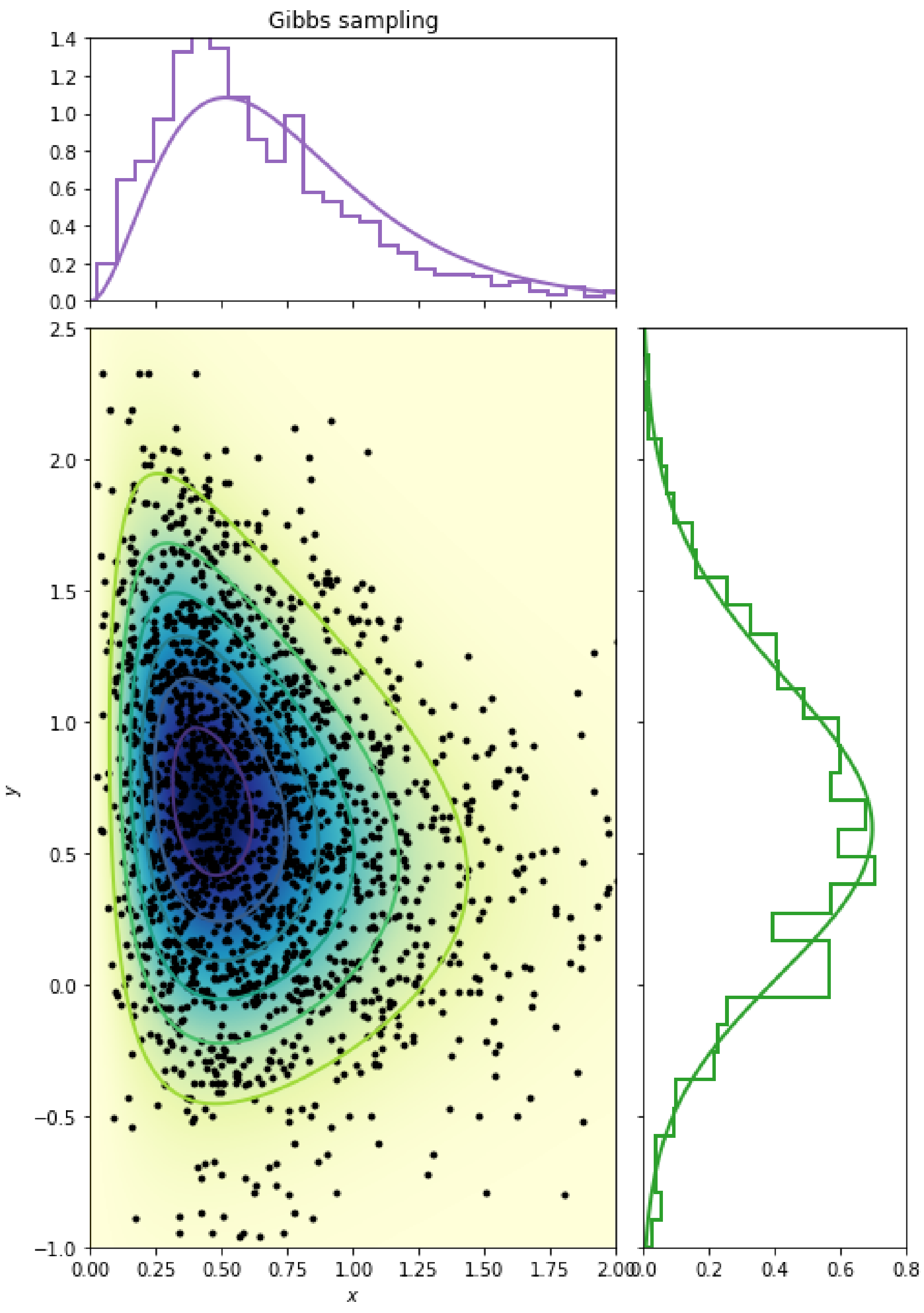
with $\Gamma(k,\theta)(t) = \frac{1}{\Gamma(k)\theta^k} t^{k-1} \mathrm{e}^{-t/\theta}$





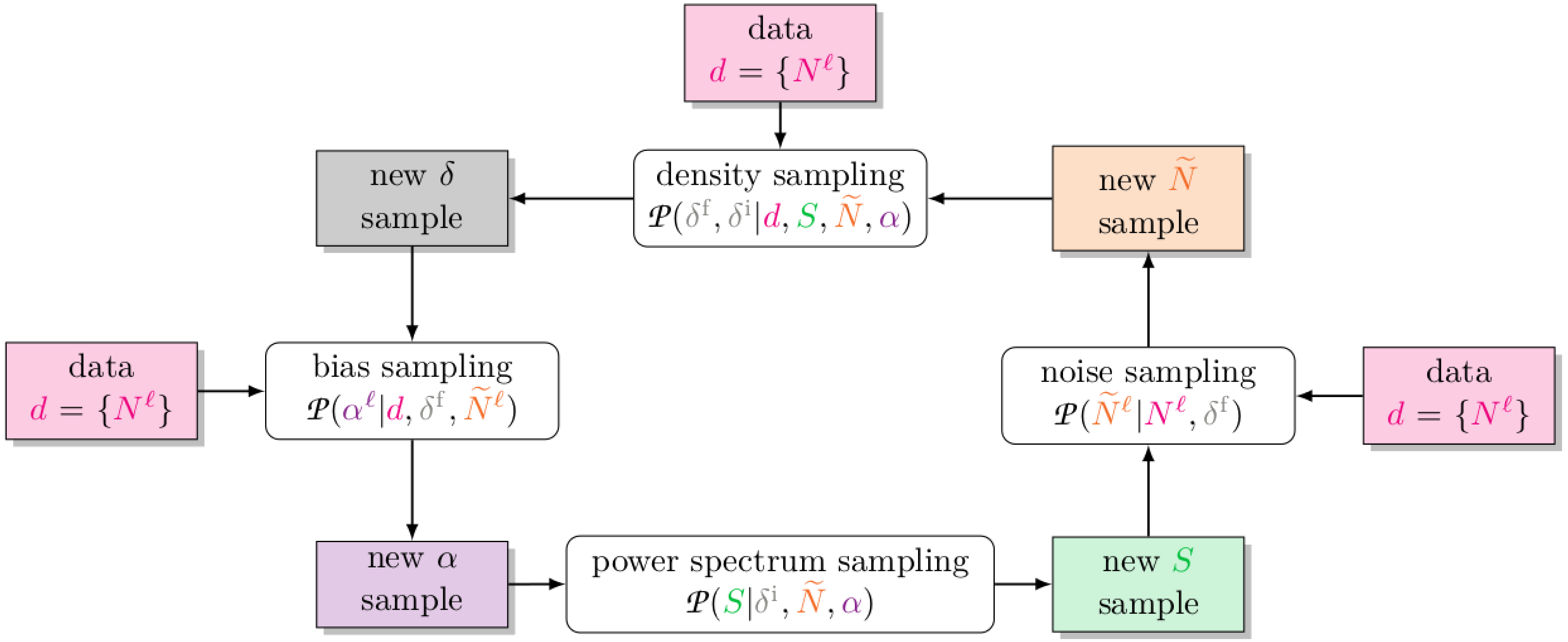
Gibbs sampler

```python
def Gibbs_sampler(target_conditional_given_x,target_conditional_given_y,Nsamp,x_start,y_start):
    x=x_start
    y=y_start
    samples_x=[x]
    samples_y=[y]
    while len(samples_x)<Nsamp-1:
        # first update x given y
        x=target_conditional_given_y(y).rvs()
        samples_x.append(x)
        samples_y.append(y)
        # then update y given x
        y=target_conditional_given_x(x).rvs()
        samples_x.append(x)
        samples_y.append(y)
    # last step, just update x given y
    x=target_conditional_given_y(y).rvs()
    # since Gibbs sampling is rejection-free,
    # here we don't even check for acceptance
    samples_x.append(x)
    samples_y.append(y)
    return samples_x, samples_y
```



Modular probabilistic programming: example

- ARES: Algorithm for Reconstruction and Sampling

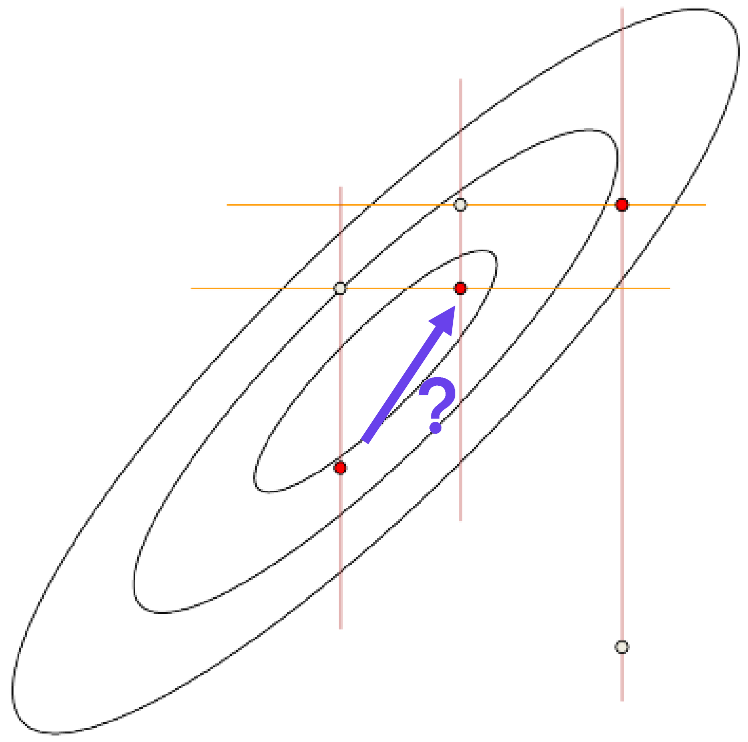

data $d = \{N^\ell\}$
density sampling $\mathcal{P}(\delta^\mathrm{f}, \delta^\mathrm{i} | d, S, \widetilde{N}, \alpha)$
new δ sample
new $\widetilde{N}$ sample
data $d = \{N^\ell\}$
bias sampling $\mathcal{P}(\alpha^\ell | d, \delta^\mathrm{f}, \widetilde{N}^\ell)$
noise sampling $\mathcal{P}(\widetilde{N}^\ell | N^\ell, \delta^\mathrm{f})$
data $d = \{N^\ell\}$
new α sample
power spectrum sampling $\mathcal{P}(S | \delta^\mathrm{i}, \widetilde{N}, \alpha)$
new S sample

Jasche, Kitaura, Wandelt & Enßlin, 0911.2493; Jasche & Wandelt, 1306.1821

MCMC beyond Metropolis-Hastings

- Shortcomings of standard Metropolis-Hastings:
  - Tuning of proposal distributions
  - Curse of dimensionality

- Gibbs sampling:
  - Uses conditionals of the target pdf
  - Inefficient if parameters are strongly correlated
  - How does one take diagonal steps in parameter space?



Hamiltonian (Hybrid) Monte Carlo

- Use classical mechanics to solve statistical problems!
  - The potential: $\psi(\mathbf{x}) \equiv -\ln p(\mathbf{x})$
  - The Hamiltonian: $H(\mathbf{x},\mathbf{p}) \equiv \frac{1}{2}\mathbf{p}^\intercal \mathbf{M}^{-1}\mathbf{p} + \psi(\mathbf{x})$

$$(\mathbf{x},\mathbf{p}) \Rightarrow \left\{ \begin{aligned} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} &= \frac{\partial H}{\partial \mathbf{p}} = \mathbf{M}^{-1}\mathbf{p} \\ \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} &= -\frac{\partial H}{\partial \mathbf{x}} = -\frac{\mathrm{d}\psi(\mathbf{x})}{\mathrm{d}\mathbf{x}} \end{aligned} \right\} \Rightarrow (\mathbf{x}',\mathbf{p}')$$

gradients of the pdf

$$a(\mathbf{x}',\mathbf{x}) = \mathrm{e}^{-(H'-H)} = 1 \longleftarrow$$

acceptance ratio unity

- HMC beats the curse of dimensionality by:
  - Exploiting gradients
  - Using conservation of the Hamiltonian

Duane *et al.* (1987), Phys. Lett. B 195, 2; Neal, 1206.1901

Hamiltonian Monte Carlo: implementation

- Usual implementation, including a Metropolis-Hastings test for numerical errors:

```
begin
    initialise $\boldsymbol{x}_{(0)}$;
    for $i = 1$ to $n$ do
        $\boldsymbol{p} \curvearrowleft \mathcal{N}(0,1)$ (normal distribution);
        $(\boldsymbol{x}^*_{(0)}, \boldsymbol{p}^*_{(0)}) = (\boldsymbol{x}_{(i-1)}, \boldsymbol{p})$;
        for $j = 1$ to $N_\mathrm{steps}$ do
            make a leapfrog move: $(\boldsymbol{x}^*_{(j-1)}, \boldsymbol{p}^*_{(j-1)}) \to (\boldsymbol{x}^*_{(j)}, \boldsymbol{p}^*_{(j)})$;
        end
        $(\boldsymbol{x}^*, \boldsymbol{p}^*) = (\boldsymbol{x}_{(N_\mathrm{steps})}, \boldsymbol{p}_{(N_\mathrm{steps})})$;
        $\alpha \curvearrowleft \mathcal{U}(0,1)$ (uniform distribution);
        if $\alpha < \min\left(1, \exp\left\{-\left[H(\boldsymbol{x}^*, \boldsymbol{p}^*) - H(\boldsymbol{x}^*_{(0)}, \boldsymbol{p}^*_{(0)})\right]\right\}\right)$ then
            $\boldsymbol{x}_{(i)} = \boldsymbol{x}^*$;
        else
            $\boldsymbol{x}_{(i)} = \boldsymbol{x}_{(i-1)}$;
        end
    end
    return $(\boldsymbol{x}_{(0)}, \dots, \boldsymbol{x}_{(n)})$;
end
```

- Setting up the Hamiltoninan

$$H(\mathbf{x}, \mathbf{p}) \equiv \frac{1}{2}\mathbf{p}^\intercal \mathbf{M}^{-1}\mathbf{p} + \psi(\mathbf{x})$$

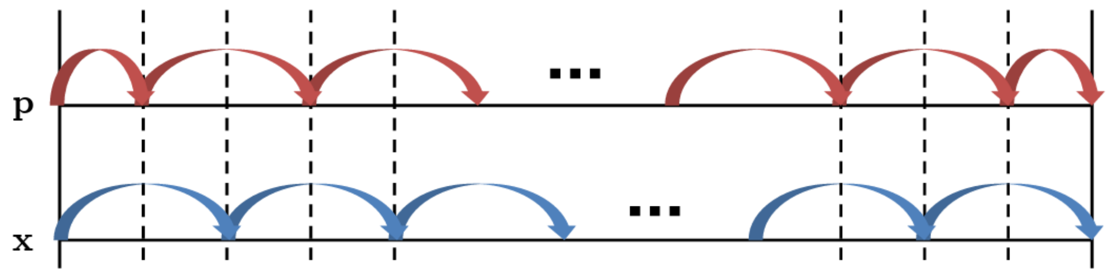
requires choosing (and tuning) a mass matrix $\mathbf{M}$.

- Equations of motion need to be discretised. The leapfrog or kick-drift-kick integrator is the usual choice because it is symplectic:

$$p_k\left(t + \frac{\epsilon}{2}\right) = p_k(t) - \frac{\epsilon}{2}\frac{\partial\psi}{\partial x_k}\bigg|_{\boldsymbol{x}(t)}$$

$$x_k\left(t + \epsilon\right) = x_k(t) + \epsilon\, p_k\left(t + \frac{\epsilon}{2}\right)$$

$$p_k\left(t + \epsilon\right) = p_k\left(t + \frac{\epsilon}{2}\right) - \frac{\epsilon}{2}\frac{\partial\psi}{\partial x_k}\bigg|_{\boldsymbol{x}(t+\epsilon)}$$



A two-dimensional test pdf

- Joint probability distribution:

$$p(x,y) \propto x^2 \exp\left(-xy^2 - y^2 + 2y - 4x\right)$$

$$\begin{cases} 0 < x < +\infty \\ -\infty < y < +\infty \end{cases}$$

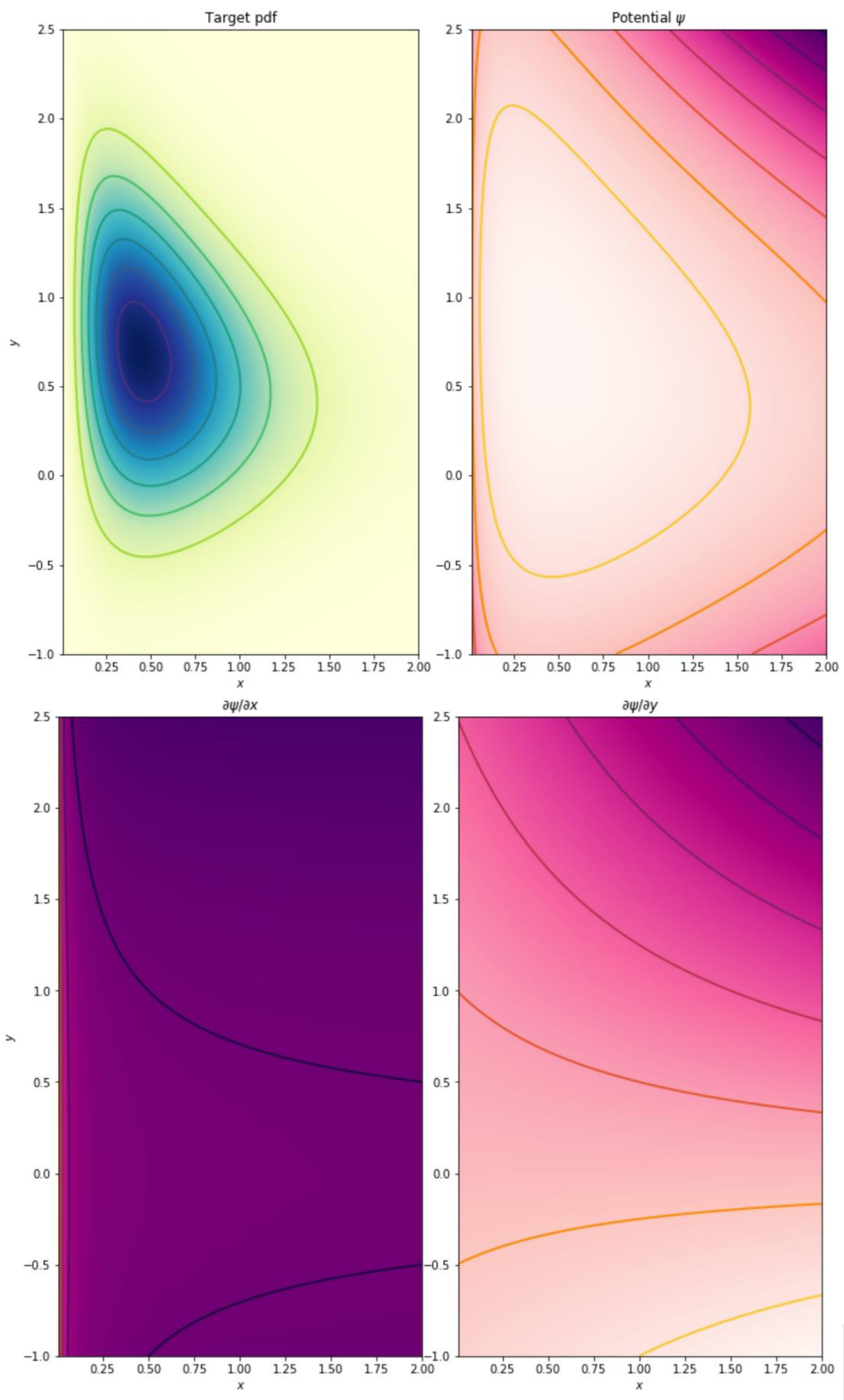
- Potential:

$$\psi(x,y) \equiv -\ln p(x,y) = -2\ln x + xy^2 + y^2 - 2y + 4x$$

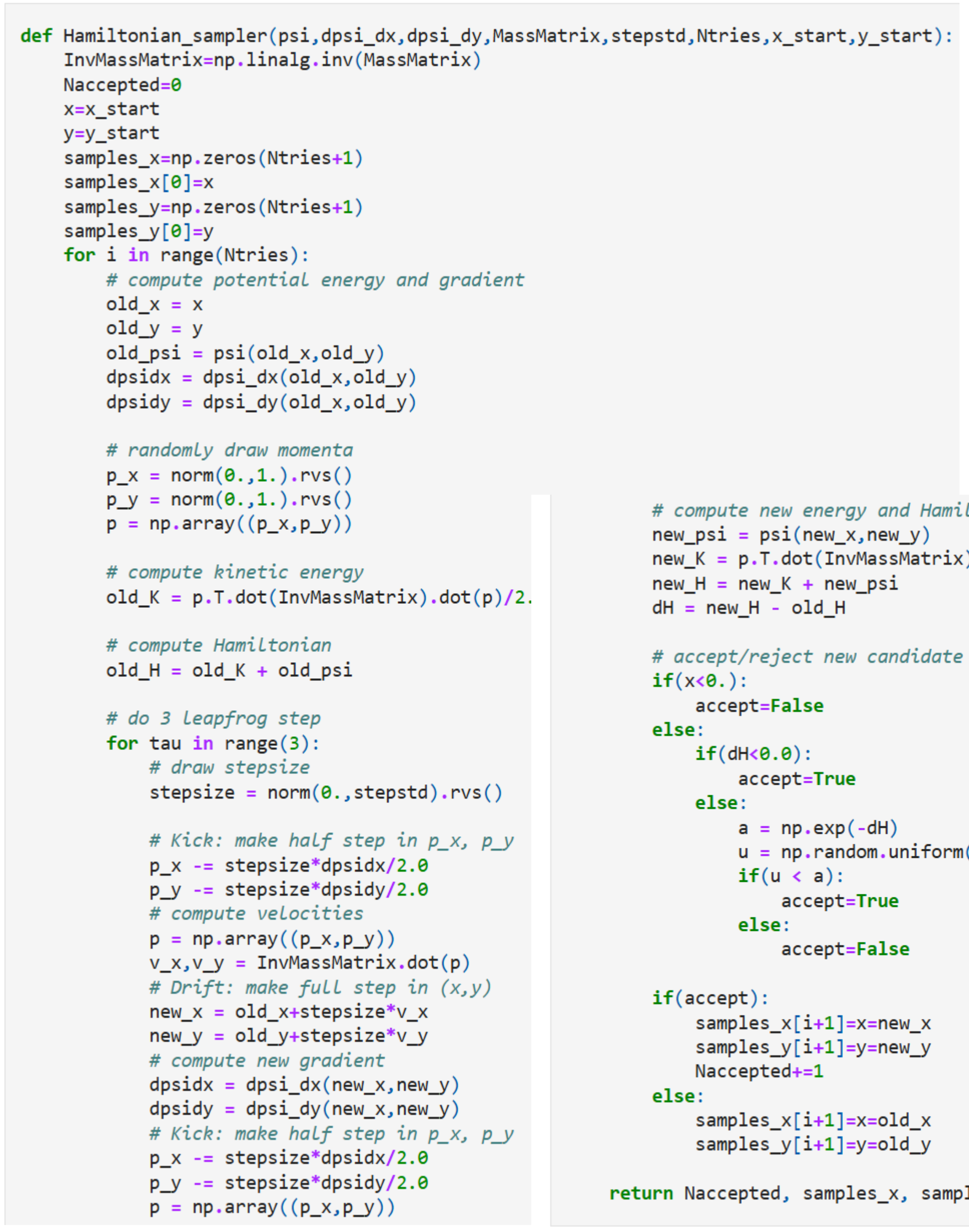
- Gradients of the potential:

$$\frac{\partial \psi}{\partial x}(x,y) = -\frac{2}{x} + y^2 + 4$$

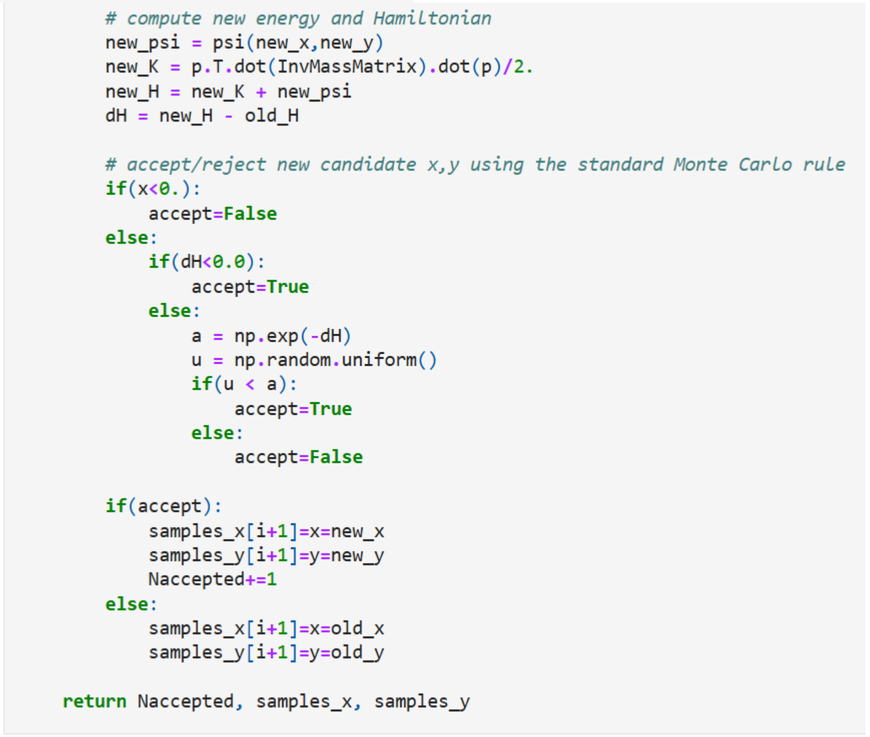
$$\frac{\partial \psi}{\partial y}(x,y) = 2xy + 2y - 2$$



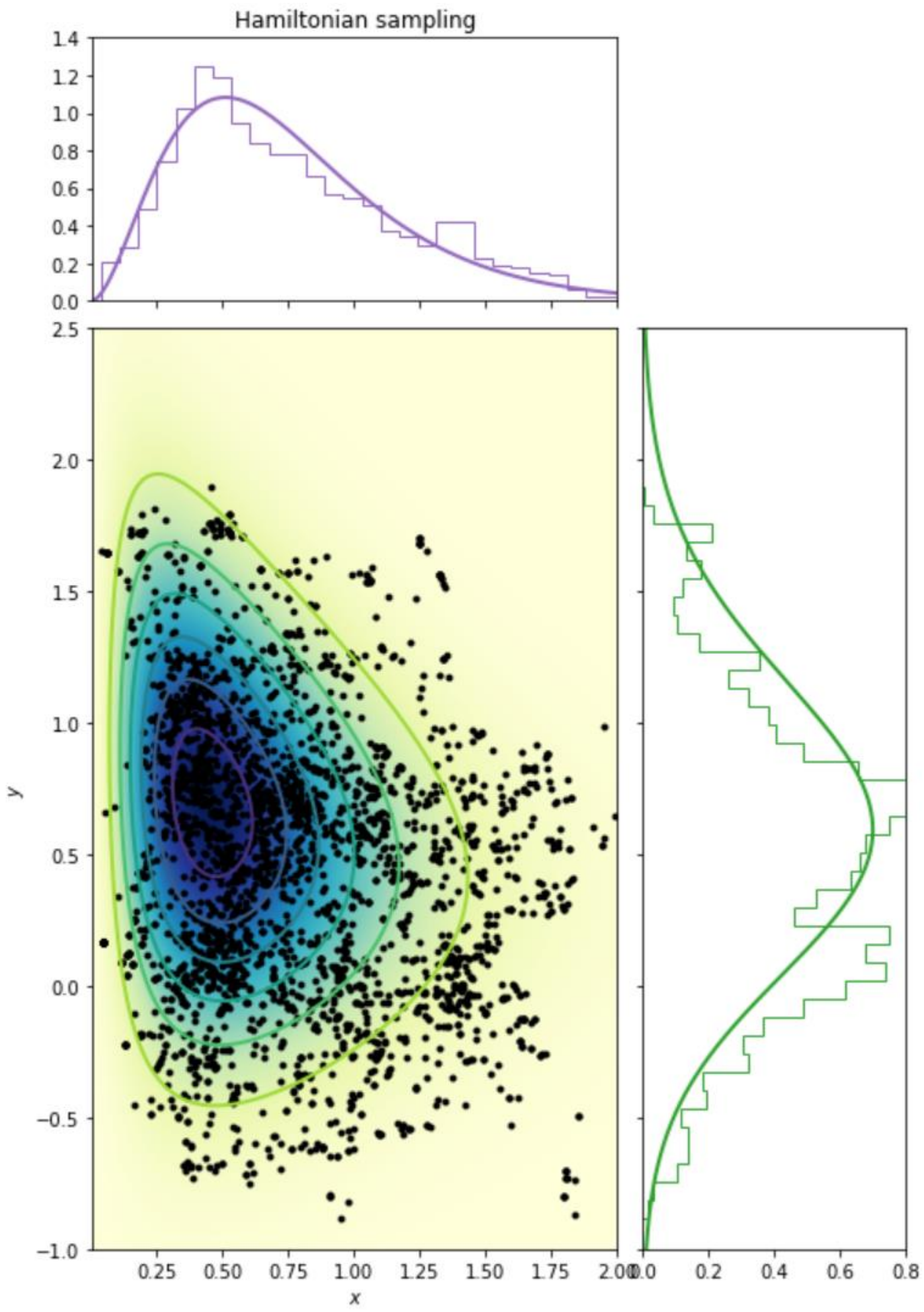
Hamiltonian sampler



```python
def Hamiltonian_sampler(psi,dpsi_dx,dpsi_dy,MassMatrix,stepstd,Ntries,x_start,y_start):
    InvMassMatrix=np.linalg.inv(MassMatrix)
    Naccepted=0
    x=x_start
    y=y_start
    samples_x=np.zeros(Ntries+1)
    samples_x[0]=x
    samples_y=np.zeros(Ntries+1)
    samples_y[0]=y
    for i in range(Ntries):
        # compute potential energy and gradient
        old_x = x
        old_y = y
        old_psi = psi(old_x,old_y)
        dpsidx = dpsi_dx(old_x,old_y)
        dpsidy = dpsi_dy(old_x,old_y)

        # randomly draw momenta
        p_x = norm(0.,1.).rvs()
        p_y = norm(0.,1.).rvs()
        p = np.array((p_x,p_y))

        # compute kinetic energy
        old_K = p.T.dot(InvMassMatrix).dot(p)/2.

        # compute Hamiltonian
        old_H = old_K + old_psi

        # do 3 leapfrog step
        for tau in range(3):
            # draw stepsize
            stepsize = norm(0.,stepstd).rvs()

            # Kick: make half step in p_x, p_y
            p_x -= stepsize*dpsidx/2.0
            p_y -= stepsize*dpsidy/2.0
            # compute velocities
            p = np.array((p_x,p_y))
            v_x,v_y = InvMassMatrix.dot(p)
            # Drift: make full step in (x,y)
            new_x = old_x+stepsize*v_x
            new_y = old_y+stepsize*v_y
            # compute new gradient
            dpsidx = dpsi_dx(new_x,new_y)
            dpsidy = dpsi_dy(new_x,new_y)
            # Kick: make half step in p_x, p_y
            p_x -= stepsize*dpsidx/2.0
            p_y -= stepsize*dpsidy/2.0
            p = np.array((p_x,p_y))
```

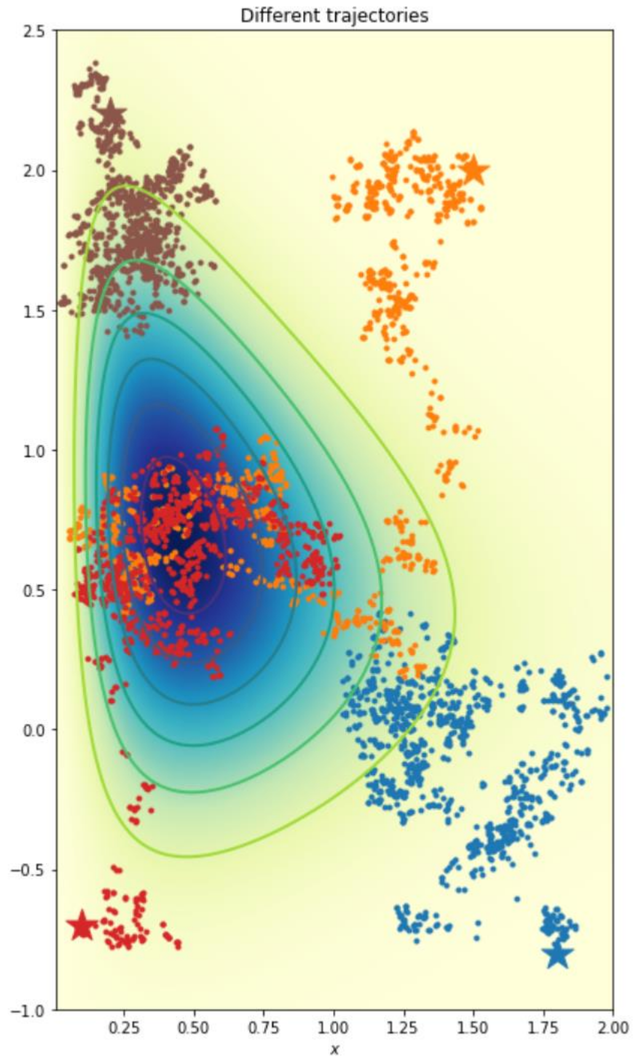


```python
        # compute new energy and Hamiltonian
        new_psi = psi(new_x,new_y)
        new_K = p.T.dot(InvMassMatrix).dot(p)/2.
        new_H = new_K + new_psi
        dH = new_H - old_H

        # accept/reject new candidate x,y using the standard Monte Carlo rule
        if(x<0.):
            accept=False
        else:
            if(dH<0.0):
                accept=True
            else:
                a = np.exp(-dH)
                u = np.random.uniform()
                if(u < a):
                    accept=True
                else:
                    accept=False

        if(accept):
            samples_x[i+1]=x=new_x
            samples_y[i+1]=y=new_y
            Naccepted+=1
        else:
            samples_x[i+1]=x=old_x
            samples_y[i+1]=y=old_y

    return Naccepted, samples_x, samples_y
```

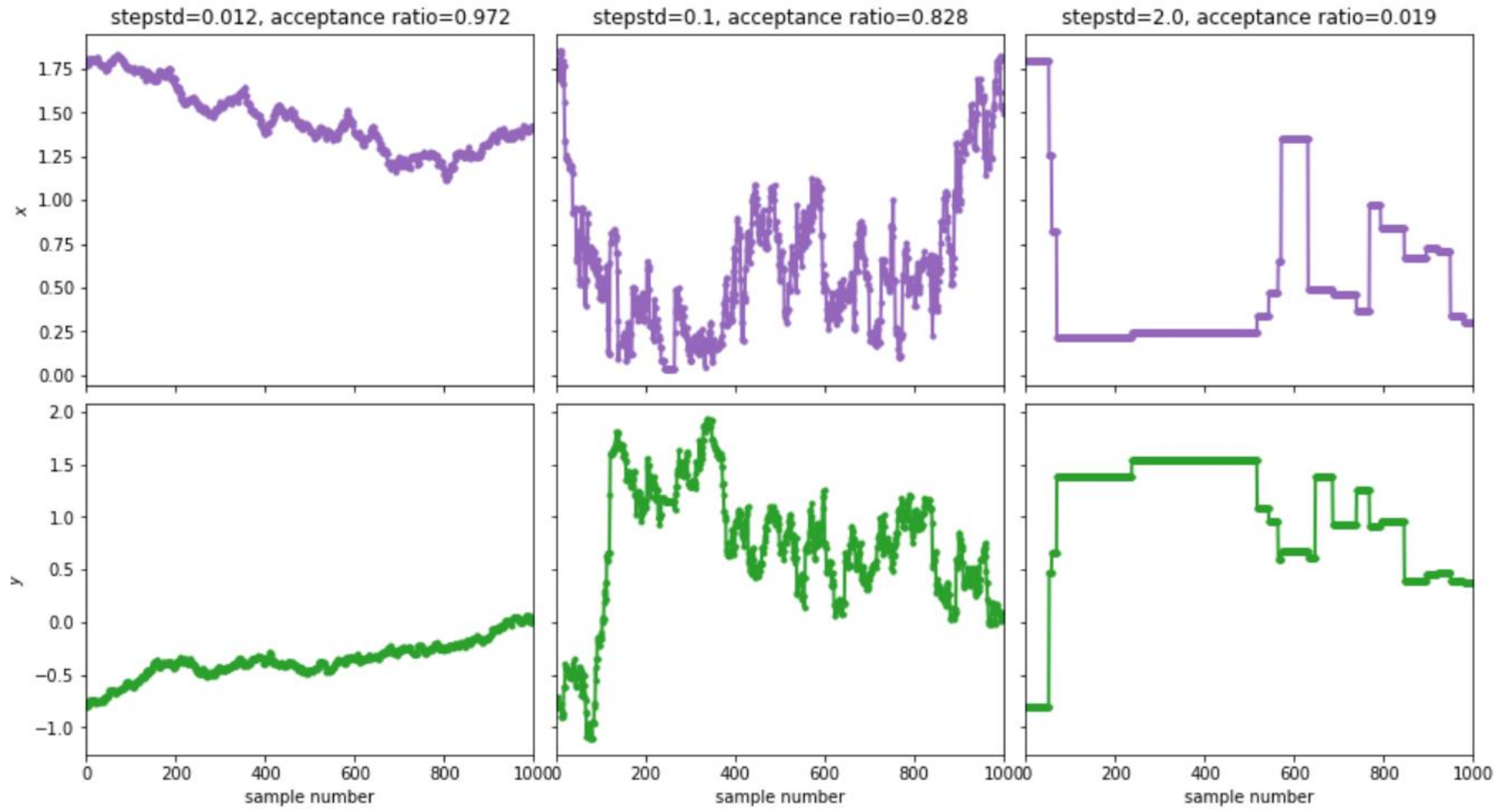


Tuning a Hamiltonian sampler

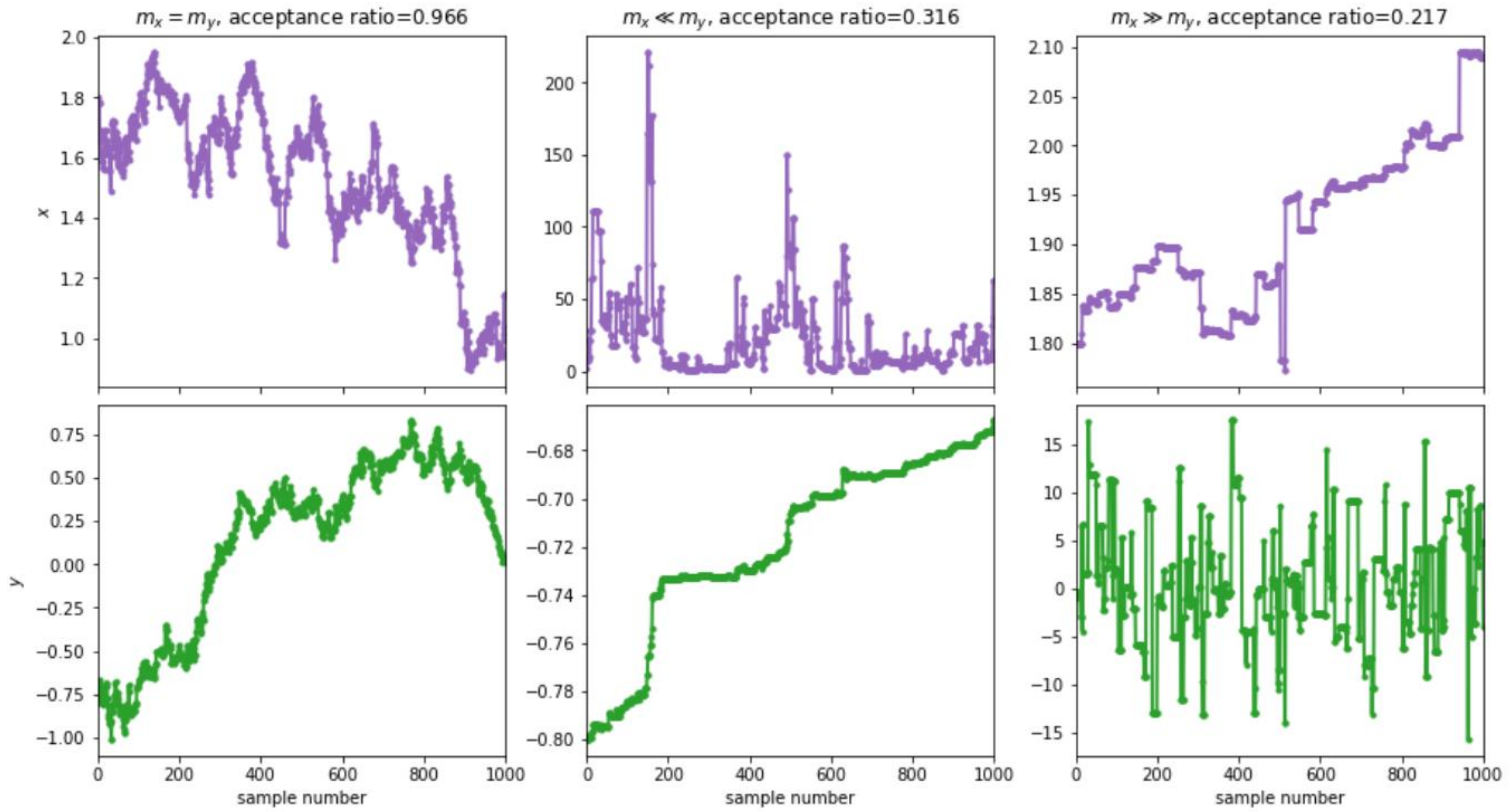
- Different trajectories



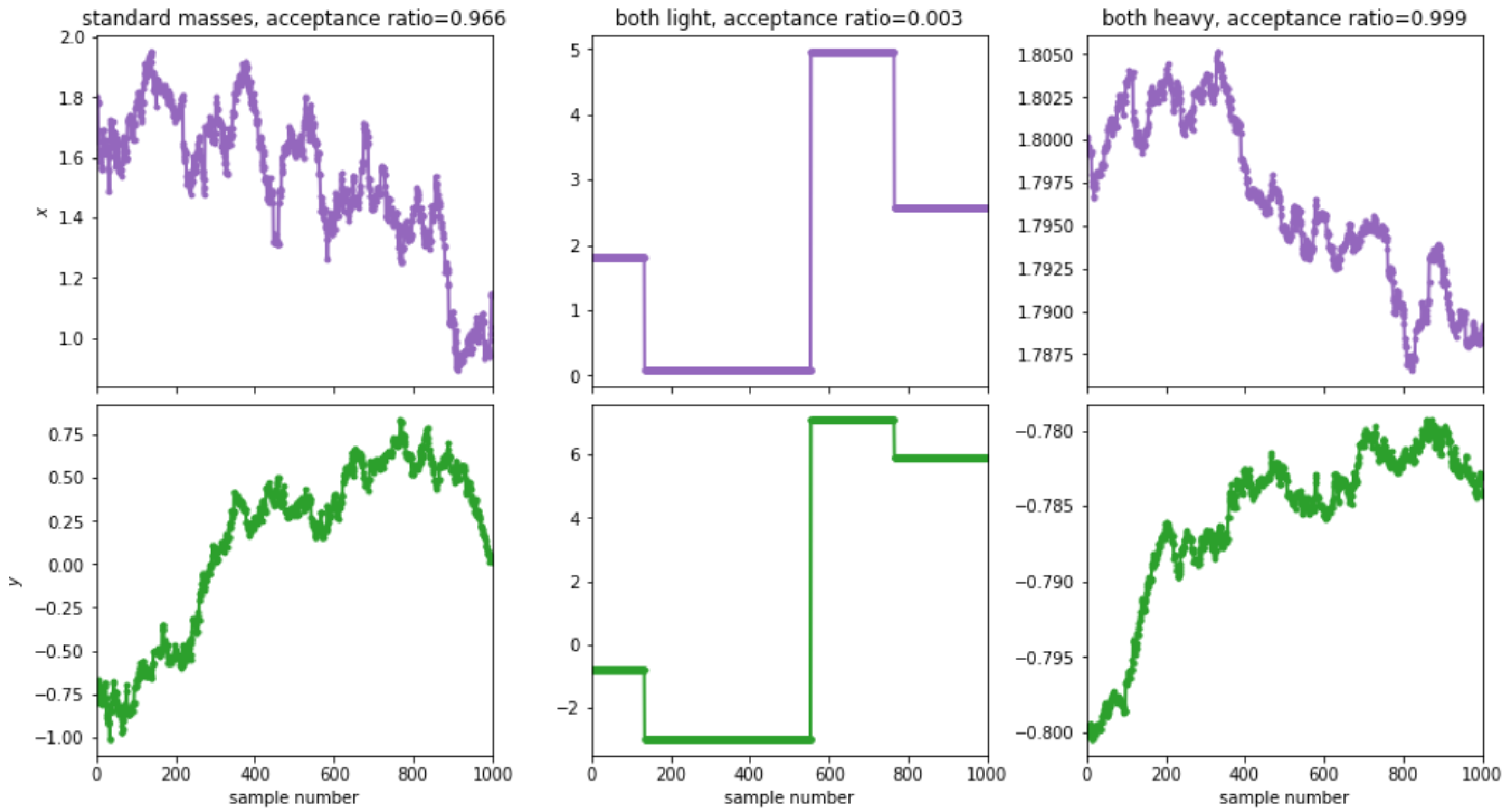
- Tuning the step size ϵ



Tuning the mass matrix



Tuning the mass matrix



References and acknowledgements

References:

- A. Gelman *et al.* (2021), *Bayesian Data Analysis, Third edition*
- C. Geyer (2011), *Introduction to Markov Chain Monte Carlo*
- R. M. Neal (2011), 1206.1901, *MCMC using Hamiltonian Dynamics*

- For his lectures, thanks to Benjamin Wandelt

https://florent-leclercq.eu/teaching.php