Advanced computational statistics, lecture 6

Frank Miller, Department of Computer and Information Science, Linköping University

May 15, 2025

LINKÖPING UNIVERSITY

Course schedule

- Topic 1: **Gradient based optimisation**
- Topic 2: **Stochastic gradient based optimisation**
- Topic 3: **Gradient free optimisation**
- Topic 4: **Optimisation with constraints**
- Topic 5: **EM algorithm and bootstrap**
- Topic 6: **Simulation of random variables**
- Topic 7: **Numerical and Monte Carlo integration; importance sampling**

Course homepage: http://www.adoptdesign.de/frankmillereu/adcompstat2025.html

Includes schedule, reading material, lecture notes, assignments

Simulation in Statistics

- Computer-generated random variables
- Purpose:
  - Simulate a situation where a statistical model can be assumed
  - Simulate situation repeatedly to investigate properties of estimators, confidence intervals, significance tests
    - Example: power of a test in situations where assumptions are not fulfilled
  - Perform Monte Carlo integration

- Problem: Given a density f of a target distribution, generate random draws $X_1, \ldots, X_n$ which follow the target distribution

Random variables from familiar distributions

- Computer-generated random variables are not really random but deterministic (Gentle, Härdle, Mori, 2012, Ch.3)
- Algorithms are used such that the deterministic nature is not visible, and variables seem random

- Deterministic algorithm generates values between 0 and 1 which follow well independent draws from Unif[0,1]
- Then, random variables following other familiar distributions can be generated from Unif[0,1] and are implemented in statistical software, see Givens and Hoeting (2013), Tab. 6.1

Random variables of familiar distributions in R

- In R, random variables can be generated for a number of distributions, e.g:
- **rbeta, rcauchy, rchisq, rexp, rf, rgamma, rlnorm, rnorm, rt, runif, rweibull**
- **rbinom, rgeom, rhyper, rmultinom, rnbinom, rpois**

```
x <- rnorm(6, mean = 1.2, sd = 2)
x
[1]  3.8839870  2.8328797  3.5344539 -2.5464309  3.2059822  0.1872261

rbinom(25, size = 3, prob = 0.25)
[1] 1 2 0 0 0 0 0 2 3 0 0 2 1 1 0 0 1 0 1 1 2 2 1 0 0
```

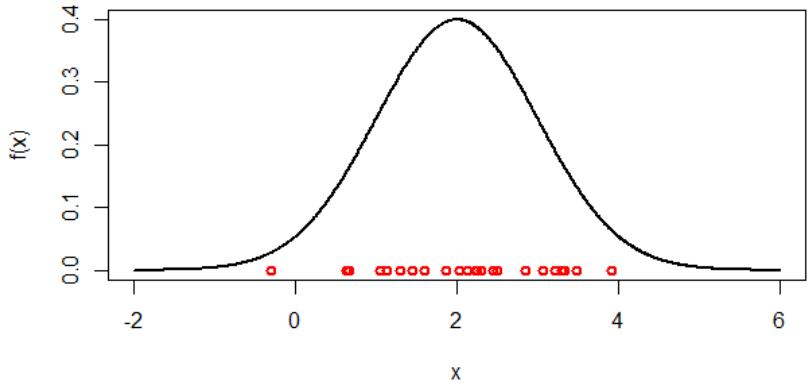
Random variables from non-familiar distributions

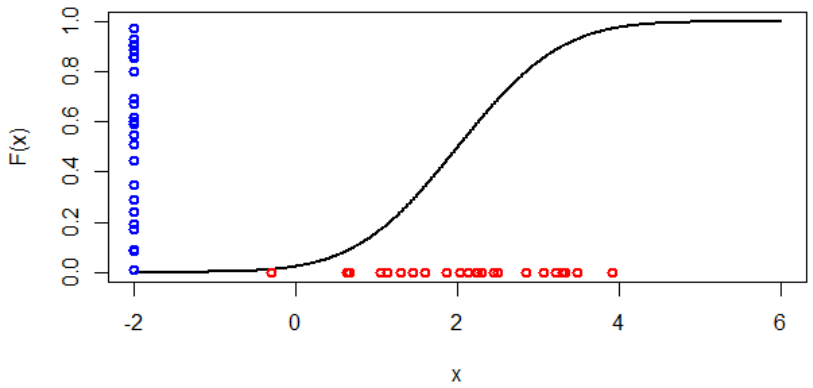
- Problem: Given a density f of a target distribution, generate random draws $X_1, \ldots, X_n$ which follow the target distribution
- Now: Density f of arbitrary form

- Methods we will consider:
  - Inverse transformation method
  - Rejection sampling
  - Composition sampling
  - Sampling importance resampling (SIR)
  - Markov chain Monte Carlo (MCMC)

Inverse transformation method

- Continuous random variable X with density f and distribution function F
- Then: $F(X)$ is uniformly distributed on [0,1]





- Therefore: if we can generate uniformly distributed random variables U, we can compute $X = F^{-1}(U)$ and obtain the desired sample

Inverse transformation method

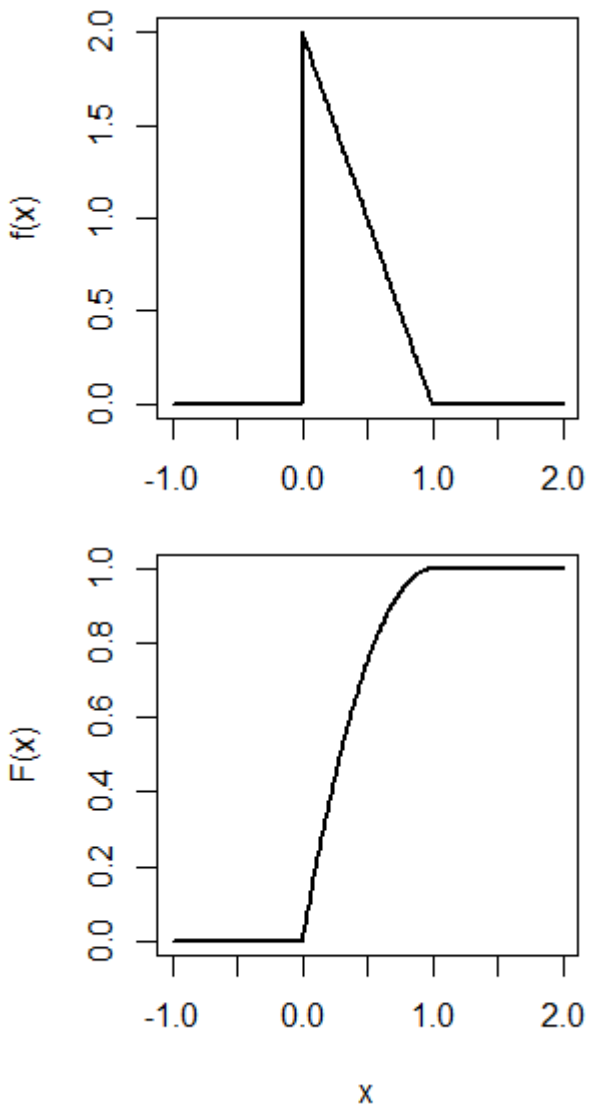
- Example 1: We want to generate random variables X with triangle distribution having density

$$f(x) = \begin{cases} 2 - 2x, & \text{if } 0 \le x \le 1, \\ 0, & \text{otherwise} \end{cases}$$

- We compute the distribution function:

$$F(x) = \int_{-\infty}^{x} f(t)\,dt = \begin{cases} 0, & \text{if } x < 0, \\ 2x - x^2, & \text{if } 0 \le x \le 1, \\ 1, & \text{if } x > 1. \end{cases}$$

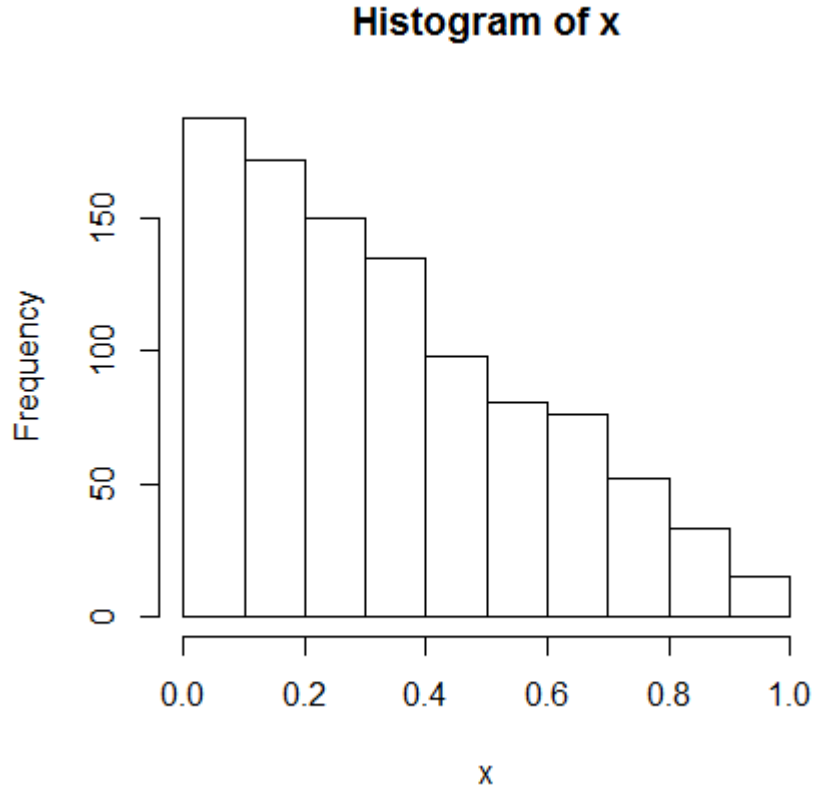
- The inverse distribution function is
  $F^{-1}(y) = 1 - \sqrt{1 - y}$
  since $y = 2x - x^2 \;\Leftrightarrow\; x^2 - 2x + y = 0 \;\Leftrightarrow$
  $x_{1,2} = 1 \pm \sqrt{1 - y} \;\Rightarrow 1 - \sqrt{1 - y}$



Inverse transformation method

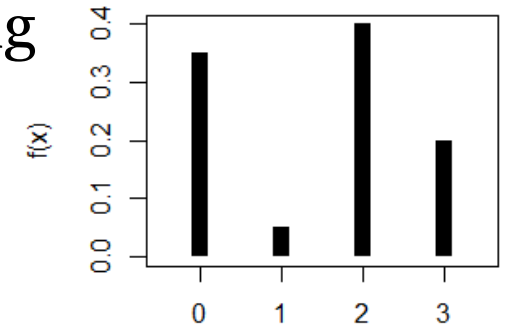
- 1000 random numbers for the triangle distribution can be generated by:

```
u <- runif(1000)
x <- 1-sqrt(1-u)
hist(x)
```

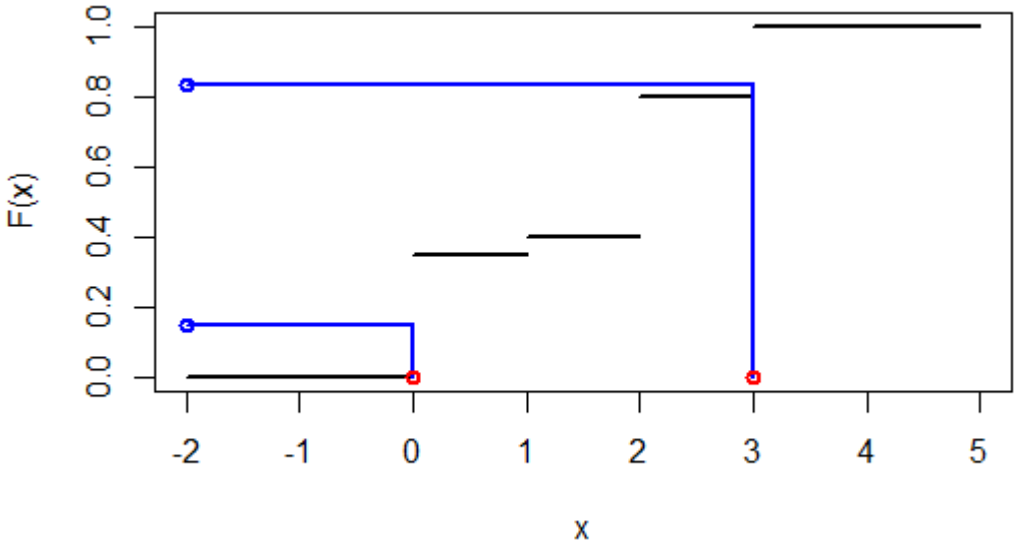


Inverse transformation – discrete random variables

- Example 2: We want to generate a random variable X being
  0 with probability 0.35,
  1 with probability 0.05,
  2 with probability 0.4,
  3 with probability 0.2

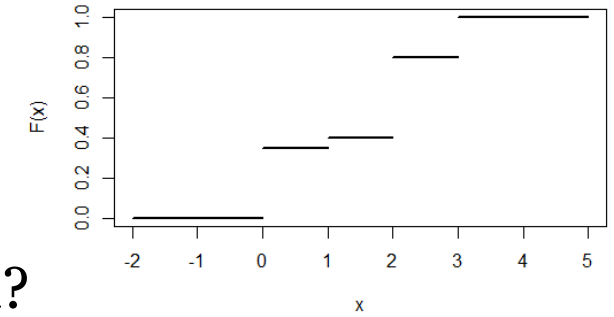


- $F(x) = P(X \leq x)$; how to apply the inverse transformation method?



Inverse transformation - discrete random variables

- Example 2: We want to generate a random variable X being
  0 with probability 0.35,
  1 with probability 0.05,
  2 with probability 0.4,
  3 with probability 0.2



- How to apply the inverse transformation method?
- Generate $U \sim \text{Unif}[0,1]$
- If $U \leq 0.35$, then $X = 0$,
  if $0.35 < U \leq 0.4$, then $X = 1$,
  if $0.4 < U \leq 0.8$, then $X = 2$,
  if $0.8 < U$, then $X = 3$.

```
u <- runif(100000)
x <- (u>0.35)+(u>0.4)+(u>0.8)
```

This is 1 if the condition in (...) is true, otherwise it is 0

Python, Julia, Matlab:

```
x = (u > 0.35).astype(int) + (u > 0.4).astype(int) + (u > 0.8).astype(int)
x = (u .> 0.35) .+ (u .> 0.4) .+ (u .> 0.8)
x = (u > 0.35) + (u > 0.4) + (u > 0.8);
```

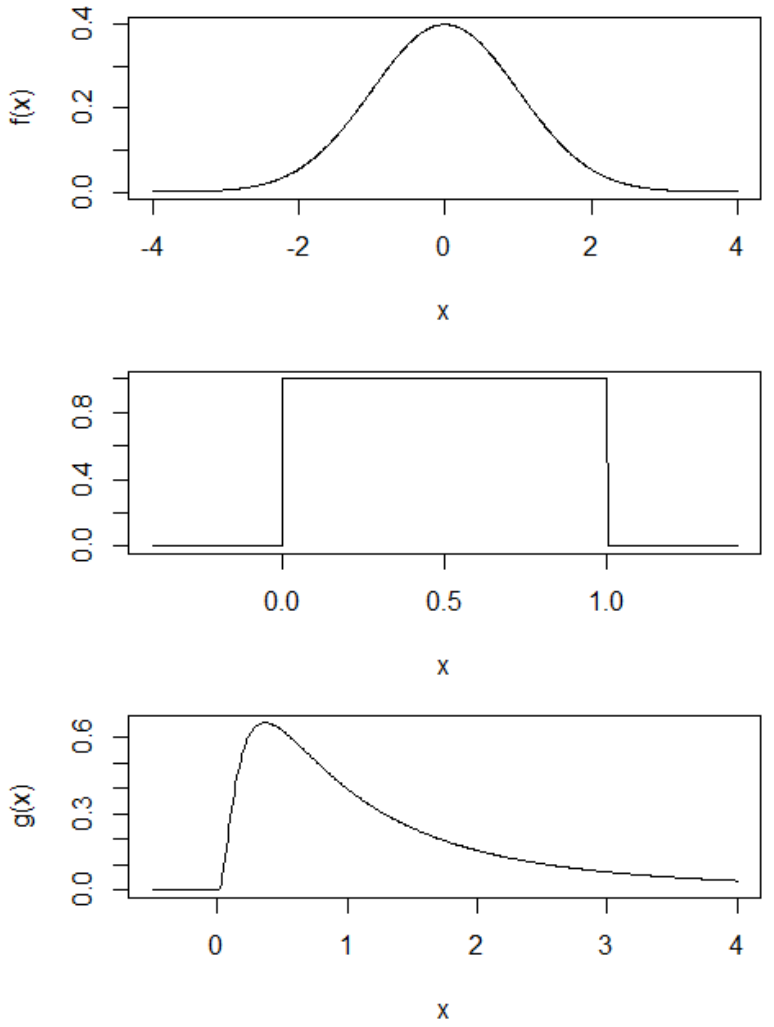
Inverse transformation method

- Inverse transformation worked well in preceding examples
- In general, drawbacks are:
  - Computation of F^{-1} might be difficult
  - Not ~~possible~~ easy to generalize to multiple dimensions*
  - Often less efficient as alternatives

- *See next slide

Inverse transformation is optimal transport

- Optimal transport:
  - Two probability measures F and G given
  - Find a transport map T which transports the probability mass from F to G, optimal according to some loss function
  - Under some assumptions about the loss, $T(x) = G^{-1}(F(x))$ is the optimal transport map
- For F = uniform distribution, optimal transport is the inverse transformation method
- Optimal transport can be generalized to higher dimensions

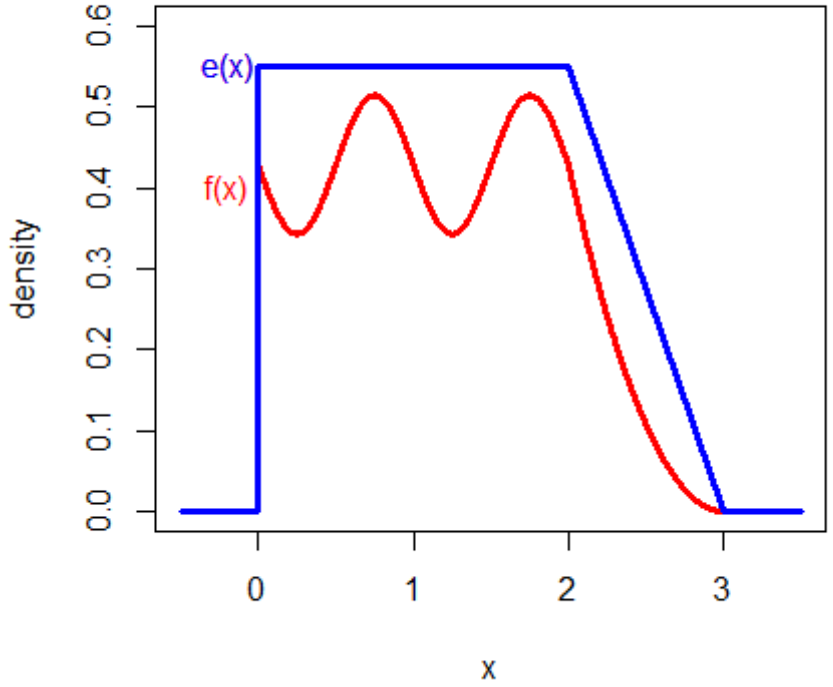


Rejection sampling

- Problem: Given a density f of a target distribution, generate random draws $X_1, \ldots, X_n$ which follow the target distribution
- It can be difficult to sample with respect to f
- Situation: There is another density g which can be sampled from and which is after scaling larger than f for all x,

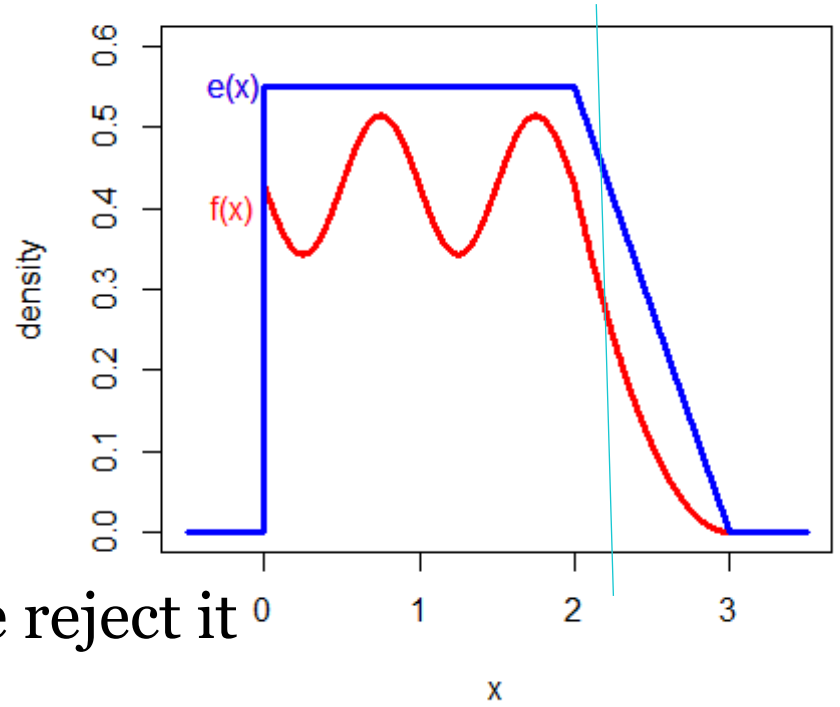
  $$e(x) = g(x)/\alpha \geq f(x)$$

  for all x and some $\alpha < 1$
- $e(x)$ is called ”envelope”



Rejection sampling

- $e(x) = g(x)/\alpha \geq f(x)$ for all x and some $\alpha < 1$



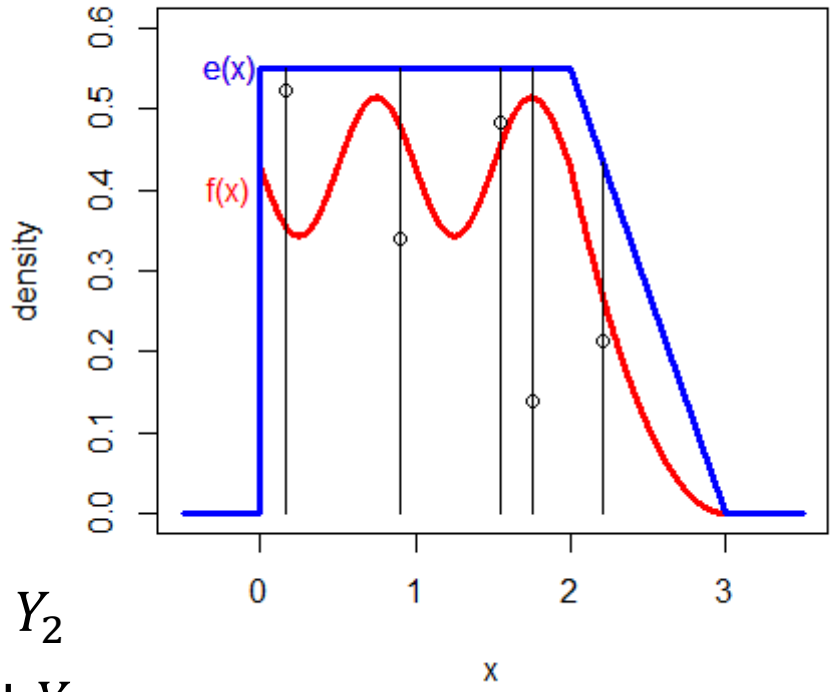
- Rejection sampling algorithm:

1. Sample $Y \sim g$
2. Sample $U \sim \text{Unif}(0,1)$
3. If $U \leq f(Y)/e(Y)$, accept Y; set $X = Y$; otherwise reject it
4. If more samples desired go to 1.

Example (for picture above): $Y = 2.21$; $f(Y) = 0.267, e(Y) = 0.435$, $f(Y)/e(Y) = 0.616$; sample U; If $U \leq 0.616$, use Y, otherwise reject it

Rejection sampling

1. Sample $Y \sim g = e\alpha$
2. Sample $U \sim \text{Unif}(0,1)$
3. If $U \leq f(Y)/e(Y)$, accept Y; set $X = Y$; otherwise rej.
4. If more samples desired, go to 1



Example (for picture above):

$(Y_1, U_1) = (2.21, 0.492)$ ➔ $U_1 < 0.616$ ➔ accept Y_1

$(Y_2, U_2) = (0.17, 0.952)$ ➔ $U_2 > f(0.17)/e(0.17)$ ➔ reject Y_2

$(Y_3, U_3) = (1.76, 0.250)$ ➔ $U_3 < f(1.76)/e(1.76)$ ➔ accept Y_3

$(Y_4, U_4) = (1.55, 0.880)$ ➔ $U_4 > f(1.55)/e(1.55)$ ➔ reject Y_4

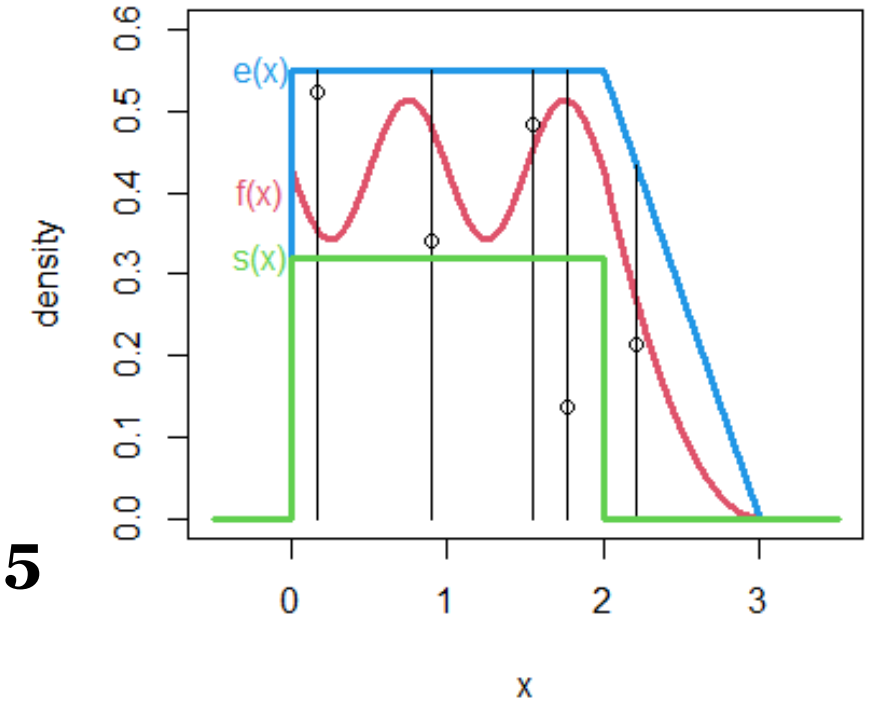
$(Y_5, U_5) = (0.90, 0.619)$ ➔ $U_5 < f(0.90)/e(0.90)$ ➔ accept Y_5

➔ use $(2.21, 1.76, 0.90)$

Squeezed rejection sampling

- $e(x) = g(x)/\alpha \geq f(x)$ for all x and some $\alpha < 1$
- **Squeezing function $s(x), s(x) \leq f(x)$**
- **Squeezed** rejection sampling algorithm:

1. Sample $Y \sim g$
2. Sample $U \sim \text{Unif}(0,1)$
3. **If $U \leq s(Y)/e(Y)$, accept Y; set $X = Y$; go to 5**
4. If $U \leq f(Y)/e(Y)$, accept Y; set $X = Y$
5. If more samples desired go to 1.



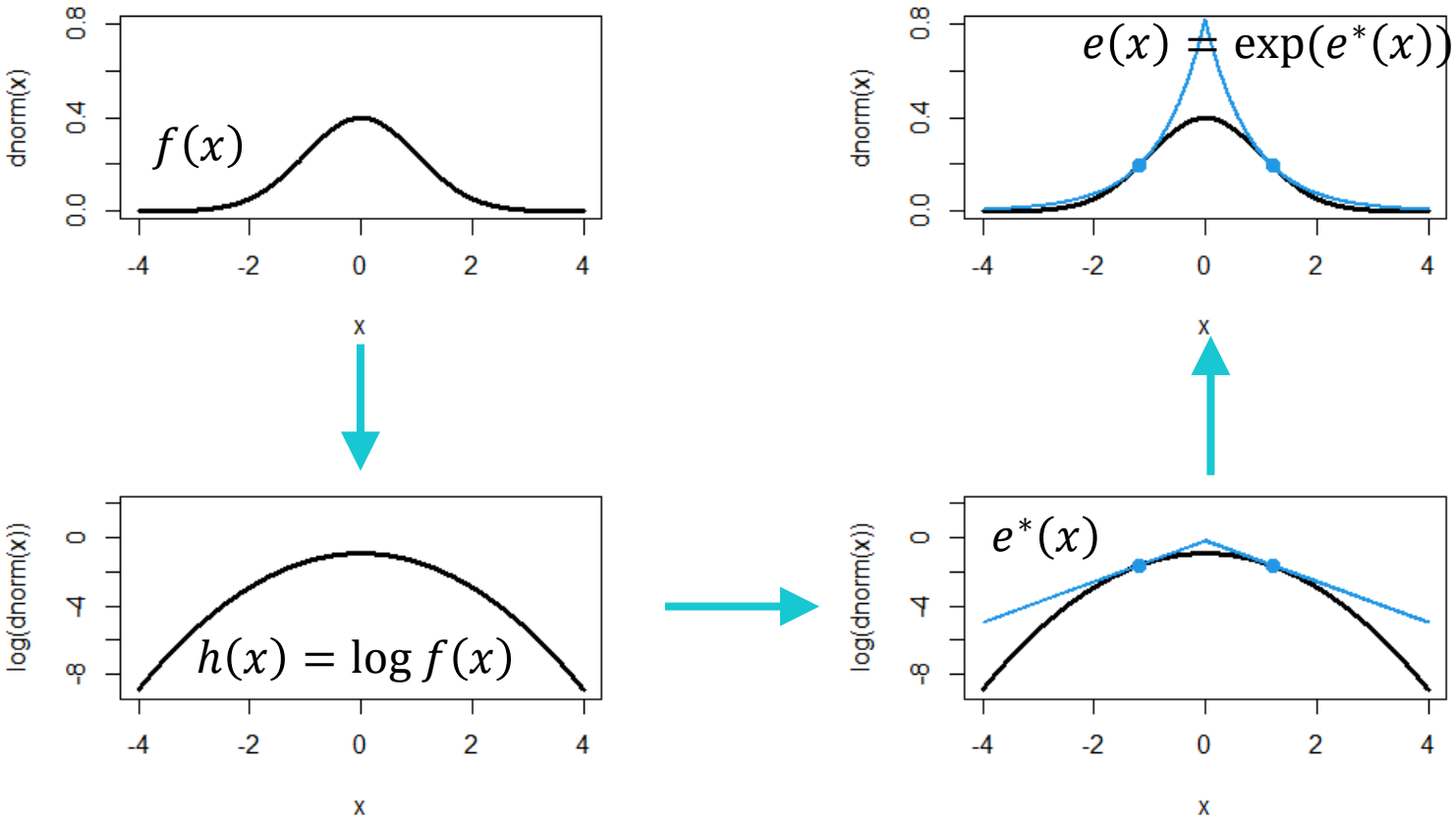
Example (for picture above): $Y = 0.90$; $s(Y) = 0.32, e(Y) = 0.55, s(Y)/e(Y) = 0.582$; sample U; If $U < 0.582$, use Y, otherwise compute $f(Y) = 0.479$, $f(Y)/e(Y) = 0.871$, and use Y if $U < 0.871$, otherwise reject

Adaptive (squeezed) rejection sampling

- Given f, we desire **automated** generation of envelope and squeezing function
- Adapt (improve) these functions where it is necessary
- Assumption: f log-concave, continuous, differentiable, $f > 0$ on an interval I
- Start with grid $T_k = \{x_1, \ldots, x_k\}$ of points on I; consider $h = \log(f)$
- The tangents of the concave h in x_i form an upper hull $e^*(x)$ of h,
  ➔ $e(x) = \exp(e^*(x))$ is an envelope
- The interpolations between the x_i forms a lower hull $s^*(x)$ of h,
  ➔ $s(x) = \exp(s^*(x))$ is a squeezing function
- Generate Y_i according to current e. If Y_i rejected or (if we squeeze) if x is accepted in Step 4 $[s(x)/e(x) < U \leq f(x)/e(x)]$, then add $x = Y_i$ to T_k ➔ T_{k+1}

Adaptive rejection sampling

- Example 3: $N(0,1)$, $f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$, $T_2 = \{x_1, x_2\}, x_2 = -x_1 = 1.2$



Proportion of waste:

$$1 - 1/\int_{-\infty}^{\infty} e(x)\,dx$$

Here: $1 - 1/1.366 = 0.268$

Adaptive rejection sampling

- An adaptive rejection sampling version exists which does not require the derivative of h (secants instead of tangents are used, see Givens and Hoeting (2013; Chapter 6.2.3.2)

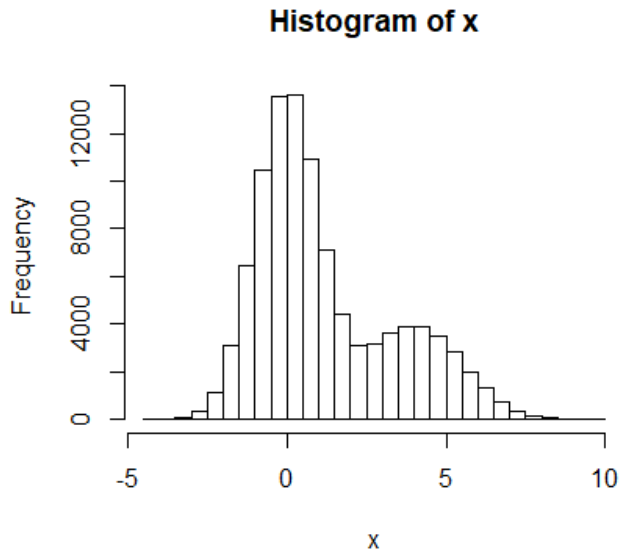
- Adaptive rejection sampling can be used for multidimensional cases, for example as subroutine in Gibbs sampling

- Many densities are log-concave, but some are not; non-log-concave densities can be handled by combining it with a Metropolis step

Composition sampling

- A finite mixture distribution can be generated by:
  - simulating the group-membership using the discrete distribution for mixing parameters
  - simulating the distribution of this group's distribution
- See Gentle, Härdle, Mori (2012), Section 3.8.7
- Ex. 4: X normal mixture of $N(0,1)$ and $N(4,1.5^2)$ with mixing parameter 0.7 and 0.3, respectively

```
g <- rbinom(100000, size = 1, prob = 0.3)
x <- rnorm(100000, mean = 4*g, sd = 1+0.5*g)
hist(x, breaks = 25)
```



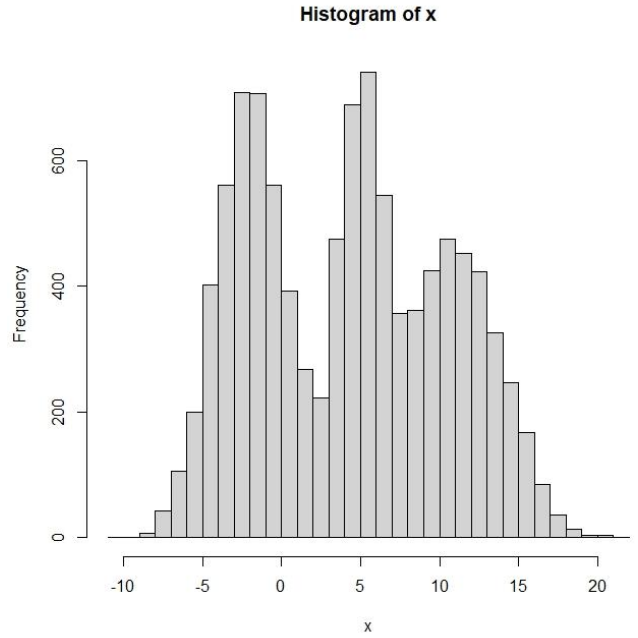
Composition sampling

- More flexible code for simulating a finite mixture distribution (e.g., a finite normal mixture) with composition sampling:
  - Define mean, standard deviations and mixing parameters as vector:

```
mu    <- c(-2, 5, 11)
sigma <- c(2.2, 1.4, 2.9)
prob  <- c(0.4, 0.25, 0.35)
n     <- 10000
```

  - Generate mixture by:

```
g <- sample(length(mu), n, replace=TRUE, p=prob)
x <- rnorm(n, mean = mu[g], sd = sigma[g])
hist(x, breaks = 25)
```



Ex. 5: Type I error of test under wrong distribution

- Given n independent and identically distributed observations $X_1, \ldots, X_n$ with mean μ, one can test H_0: $\mu = 0$ versus H_1: $\mu > 0$ with the one-sample t-test

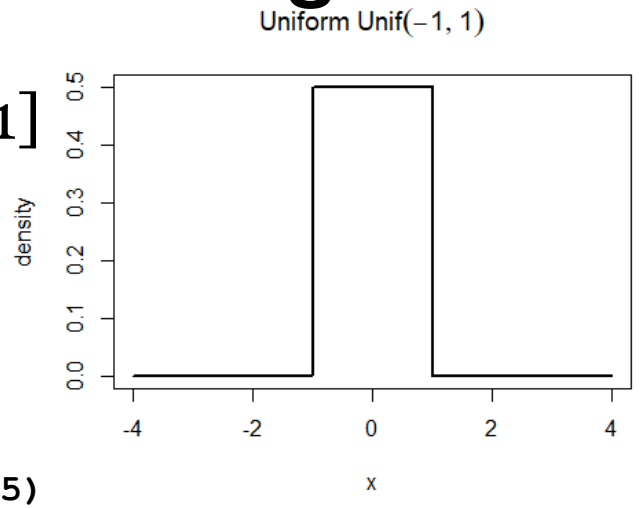
$$\text{reject } H_0 \text{ if and only if } \frac{\sqrt{n}\bar{x}}{s_x} > t_{n-1;1-\alpha}$$

- Assumption for test: normal distribution of observations
- How sensitive is t-test if observations not normal?
- We focus on H_0 first: Can type I error be larger than α (such that it matters) for certain distributions?
- Idea:
  - Choose some distributions with mean $= 0$, simulate n repetitions, perform t-test, and record if rejected
  - Repeat this s times and check rejection rate

Ex. 5: Type I error of test under wrong distribution

- For $n = 10$, simulate rejection rate for Unif[-1,1]

```
#Simulation of one sample t-test
s     <- 100000
n     <- 10
count <- 0
for (sim in 1:s)
{
  x      <- runif(n, min = -1, max = 1)
  reject <- (t.test(x, alternative = "greater")$p.value < 0.05)
  count  <- count + reject
}
#Rejection rate estimate:
rre   <- count/s
```



This is 1 if the condition in (...) is true, otherwise it is 0

- Note that there are possibilities to make simulation more efficient (e.g., by avoiding the loop) – see code on homepage

Ex. 5: Type I error of test under wrong distribution

```
s     <- 100000
n     <- 10
count <- 0
for (sim in 1:s)
{
  x      <- runif(n, min = -1, max = 1)
  reject <- (t.test(x, alternative = "greater")$p.value < 0.05)
  count  <- count + reject
}
rre   <- count/s
```

- Precision of result?

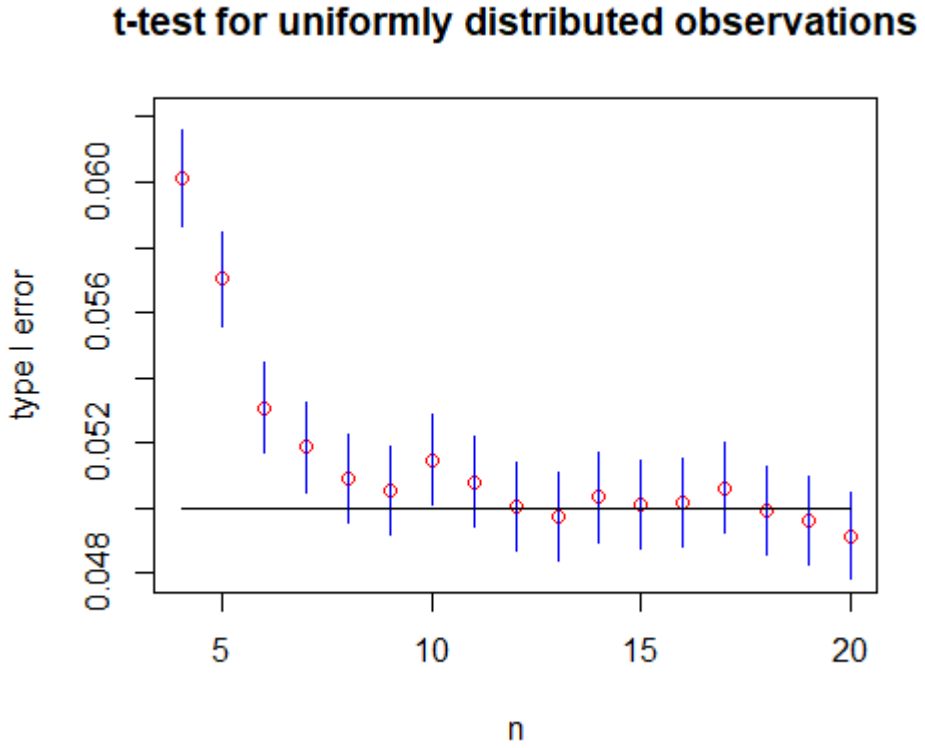
p = true rejection rate; reject$\sim Bin(1,p)$, count$\sim Bin(s = 100000, p)$

$$Var(\text{count}) = p(1-p)s, Var\left(\frac{\text{count}}{s}\right) = \frac{p(1-p)}{s}, sd(\text{rre}) = \sqrt{\frac{p(1-p)}{s}}$$

≈ 0.0007 for $p = 0.05$.

Ex. 5: Type I error of test under wrong distribution

- Simulated rejection rate for Unif[-1,1] for $n = 4, 5, \ldots, 20$ with 95%-simulation-error-CIs based on 100 000 sim. for each n
- One more loop for n used
- Took ~1 min to simulate



Ex. 5: Type I error of test under wrong distribution

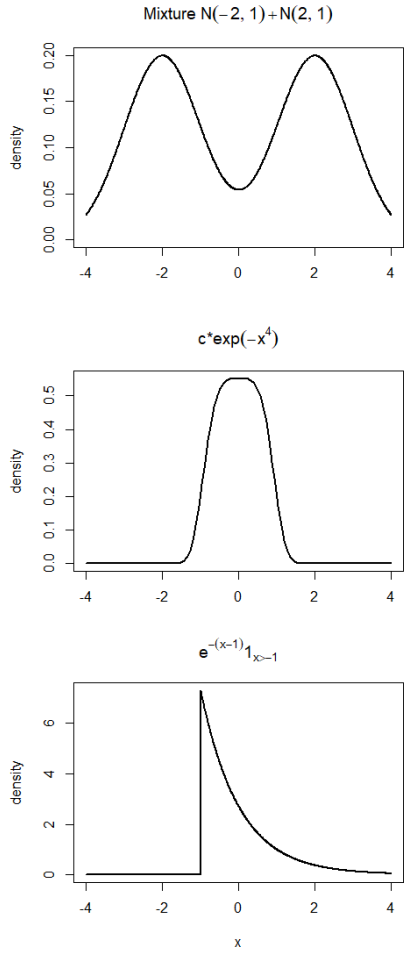
- Again, rejection rate for $n = 10$, but for:

a) An equal mixture of $N(-2,1)$ and $N(2,1)$,

b) Distribution with density: $f(x) = c\,\exp(-x^4)$,

c) Distribution with density:
$f(x) = \exp(-(x-1))\,\mathbf{1}\{x \geq -1\}$

Which simulation method in each case?



Sampling importance resampling (SIR)

- Methods considered so far generate the target distribution **exactly**
- Sampling importance resampling (SIR) is **approximate** method (this approximation is often fully ok)

- Use again envelope-function g, but do not longer require the envelope being larger than f everywhere
- If desired to draw n observations following f, start with sampling m independent observations following g (recommendation: $m \geq 10n$)
- Resample then n from these m as described below

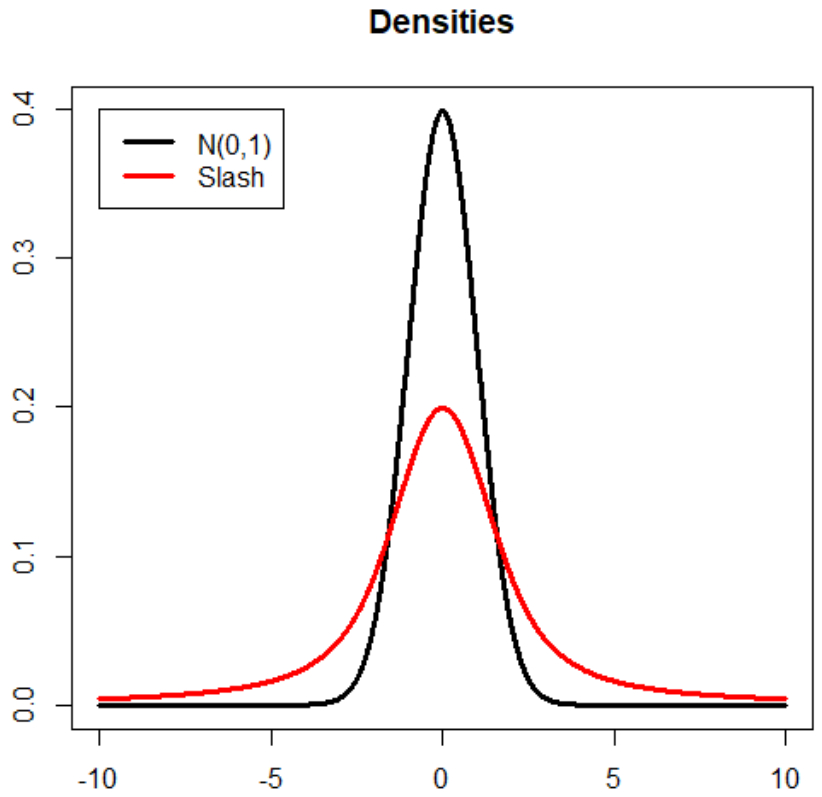
Sampling importance resampling (SIR)

1. Sample m ($\geq 10n$) random variables $Y_1, \ldots, Y_m$ from g
2. Calculate standardized importance weights
$$w(Y_i) = \frac{f(Y_i)/g(Y_i)}{\sum_{j=1}^{m} f(Y_j)/g(Y_j)}$$
for all m random draws Y_i from g.
3. Resample $X_1, \ldots, X_n$ from $Y_1, \ldots, Y_m$ with replacement with probabilities $w(Y_1), \ldots, w(Y_m)$

- $X_1, \ldots, X_n$ follow then approximately f
- Note: f need to be known only up to a constant (constant cancels out in calculation of standardized weights)

Example 6: The slash distribution

- A random variable Y has slash distribution if $Y = X/U$ with $X \sim N(0,1)$ and $U \sim \text{Unif}(0,1)$ independently



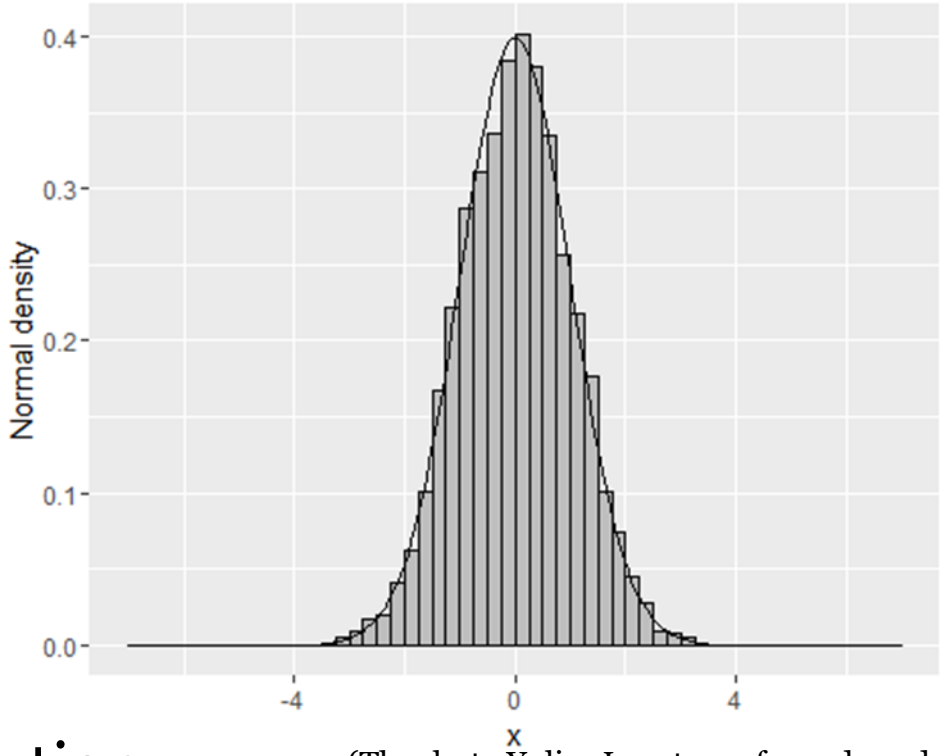
Sampling importance resampling (SIR) – Illustration

- Example 6: Use slash distribution as SIR envelope g to generate random variables following standard normal density f

```
library("extraDistr") # used for computation of slash
                      # density and simulation

sir <- function(m, n)
{
  # m - sample size from the envelope distribution
  # n - resample size
  # relative to n, m should be large
  y <- rslash(m) # sample candidates Y1,...Ym iid from g
  w <- dnorm(y)/dslash(y)
  w <- w/sum(w) # calculate the standardized weights
  x <- sample(y, n, replace=TRUE, prob=w) # resample with
                                          # probabilities = w

  return(x)
}
x <- sir(100000, 5000)
```

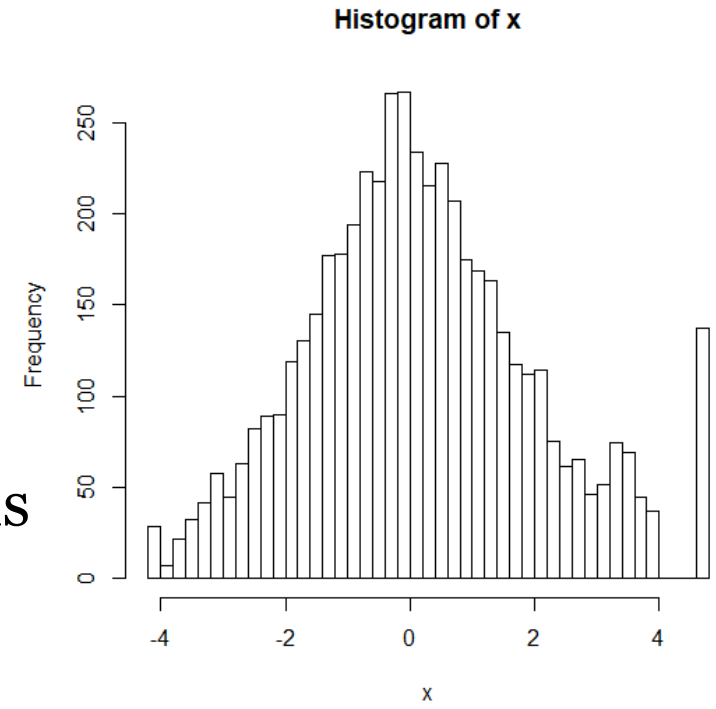


The simulated data follows well a normal distribution

(Thanks to Yuliya Leontyeva for code and illustration on this slide!)

Sampling importance resampling (SIR) – illustration

- Method worked well since envelope (slash distribution) had heavier tails than target distribution (standard normal)
- If we run SIR to generate the slash distribution with standard normal as envelope, no observations are generated at tails
- Lowest and highest values in Y-sample receive high weights (overrepresentation in X-sample)
- **Recommendation:** Use envelopes with heavier tails (or equally heavy) than the target distribution



Markov chain Monte Carlo (MCMC), see GH 7.1, 7.3

- The algorithms considered so far generate sequences of **independent** observations which follow the target distribution exactly or approximately (sampling importance resampling)
- We will now consider a method which generates a sequence of **dependent** observations which follow the target distribution approximately

- The next observation $(t+1)$ will be generated based on a proposal distribution g which depends on the current observation (t), i.e. $g(\cdot\,|X^{(t)})$
- Since $X^{(t+1)}$ depends on $X^{(t)}$ but not on earlier observations, the sequence $(X^{(t)})$ is a Markov chain

MCMC – Metropolis-Hastings algorithm

- A general method to generate the Markov chain is the Metropolis-Hastings (MH) algorithm
- A starting value $\boldsymbol{x}^{(0)}$ is generated from some starting distribution
- Given observation $\boldsymbol{x}^{(t)}$, generate $\boldsymbol{x}^{(t+1)}$ as follows:

1. Sample a candidate $\boldsymbol{x}^*$ from a proposal distribution $g(\cdot\,|\boldsymbol{x}^{(t)})$
2. Compute the MH ratio $R\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^*\right) = \dfrac{f(\boldsymbol{x}^*)\, g\left(\boldsymbol{x}^{(t)}\middle|\boldsymbol{x}^*\right)}{f\left(\boldsymbol{x}^{(t)}\right) g\left(\boldsymbol{x}^*\middle|\boldsymbol{x}^{(t)}\right)}$
3. Sample $\boldsymbol{x}^{(t+1)}$ according to

$$\boldsymbol{x}^{(t+1)} = \begin{cases} \boldsymbol{x}^*, \text{with probability } \min\{R\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^*\right), 1\} \\ \boldsymbol{x}^{(t)}, \text{otherwise} \end{cases}$$

4. If more observations needed, set `t <- t+1`; go to 1

Metropolis algorithm
Special case when g is symmetric:
$g\left(\boldsymbol{x}^*\middle|\boldsymbol{x}^{(t)}\right) = g\left(\boldsymbol{x}^{(t)}\middle|\boldsymbol{x}^*\right)$

$= \dfrac{f(\boldsymbol{x}^*)}{f\left(\boldsymbol{x}^{(t)}\right)}$

Simulated annealing

- Start value $\boldsymbol{x}^{(0)}$; stage $j = 0, 1, 2, \ldots$ has m_j iterations; initial temperature τ_0; set $j = 0$
- Given iteration $\boldsymbol{x}^{(t)}$, generate $\boldsymbol{x}^{(t+1)}$ as follows:

1. Sample a candidate $\boldsymbol{x}^*$ from a proposal distribution $p(\cdot\,|\boldsymbol{x}^{(t)})$
2. Compute $h(\boldsymbol{x}^{(t)}, \boldsymbol{x}^*) = \exp(\frac{g(\boldsymbol{x}^*) - g(\boldsymbol{x}^{(t)})}{\tau_j})$
3. Define next iteration $\boldsymbol{x}^{(t+1)}$ according to

$$\boldsymbol{x}^{(t+1)} = \begin{cases} \boldsymbol{x}^*, \text{with probability } \min\{h(\boldsymbol{x}^{(t)}, \boldsymbol{x}^*), 1\} \\ \boldsymbol{x}^{(t)}, \text{otherwise} \end{cases}$$

4. Set `t <- t+1` and repeat 1.-3. m_j times
5. Update $\tau_j = \alpha(\tau_{j-1})$ and $m_j = \beta(m_{j-1})$; set `j <- j+1`; go to 1

τ_j is temperature; function α should slowly decrease it; function β should be increasing

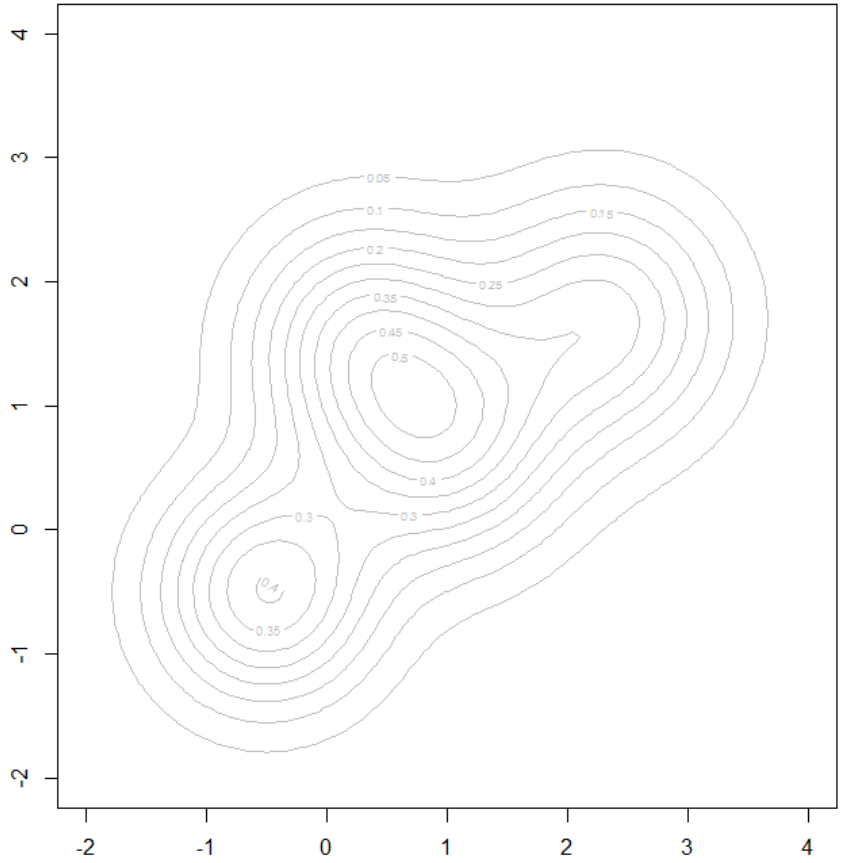
$g(\boldsymbol{x}^{(t)}) - g(\boldsymbol{x}^*)$ for minimisation

Simulated annealing and Metropolis algorithm

- For fixed temperature τ, simulated annealing algorithm is a Metropolis algorithm
- Kirkpatrick et al. (1983) proposed name simulated annealing for using it as optimisation method
- $h\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{*}\right) = \exp\left(\frac{g\left(\boldsymbol{x}^{(t)}\right) - g(\boldsymbol{x}^{*})}{\tau_j}\right) = \frac{\exp\left(-\frac{g(\boldsymbol{x}^{*})}{\tau_j}\right)}{\exp\left(-\frac{g\left(\boldsymbol{x}^{(t)}\right)}{\tau_j}\right)} = \frac{f(\boldsymbol{x}^{*})}{f\left(\boldsymbol{x}^{(t)}\right)} = R(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{*})$
- Key ingredient of Metropolis and simulated annealing alg.: Markov chain $\boldsymbol{x}^{(t)}$ **has limiting stationary distribution $\boldsymbol{f}$**; for a proof see e.g. Koski (2009)
- Requirement for all: $\boldsymbol{x}^{(t)}$ irreducible and aperiodic chain

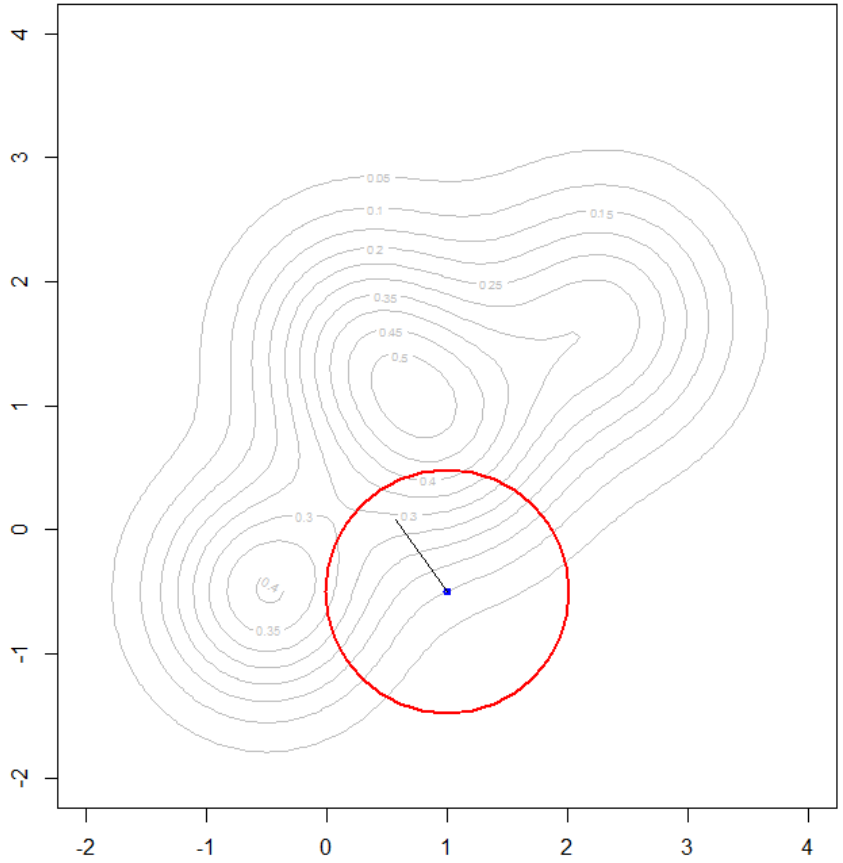
Metropolis alg. – Ex.7

- For illustration, we consider two-dimensional distribution with density f according to contour lines in figure (extended **example from L3**)
- Proposal distribution
  $g\left(\boldsymbol{x}^* \middle| \boldsymbol{x}^{(t)}\right) = g\left(\boldsymbol{x}^{(t)} \middle| \boldsymbol{x}^*\right)$
  $= \frac{1}{\pi r^2} \mathbf{1}\{\left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\| < r\}$
  for some constant r (here=1)



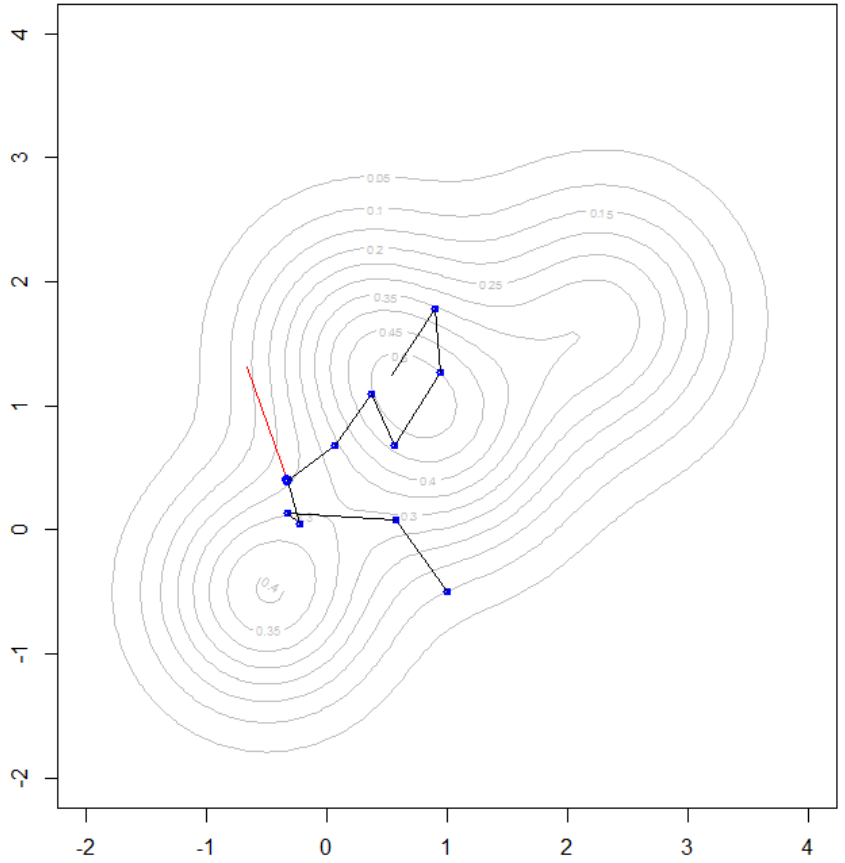
Metropolis alg. – Ex.7

- Proposal distribution
  $g(\boldsymbol{x}^*|\boldsymbol{x}^{(t)}) = g(\boldsymbol{x}^{(t)}|\boldsymbol{x}^*)$
  $= \frac{1}{\pi r^2}\mathbf{1}\{\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\| < r\}$
  for some constant r (here=1)
- Start here with $\boldsymbol{x}^{(0)}$=(1,-0.5)
- Randomize uniformly on unit circle around $\boldsymbol{x}^{(0)}$ (proposal distribution); result $\boldsymbol{x}^*$ =(0.58,0.08)



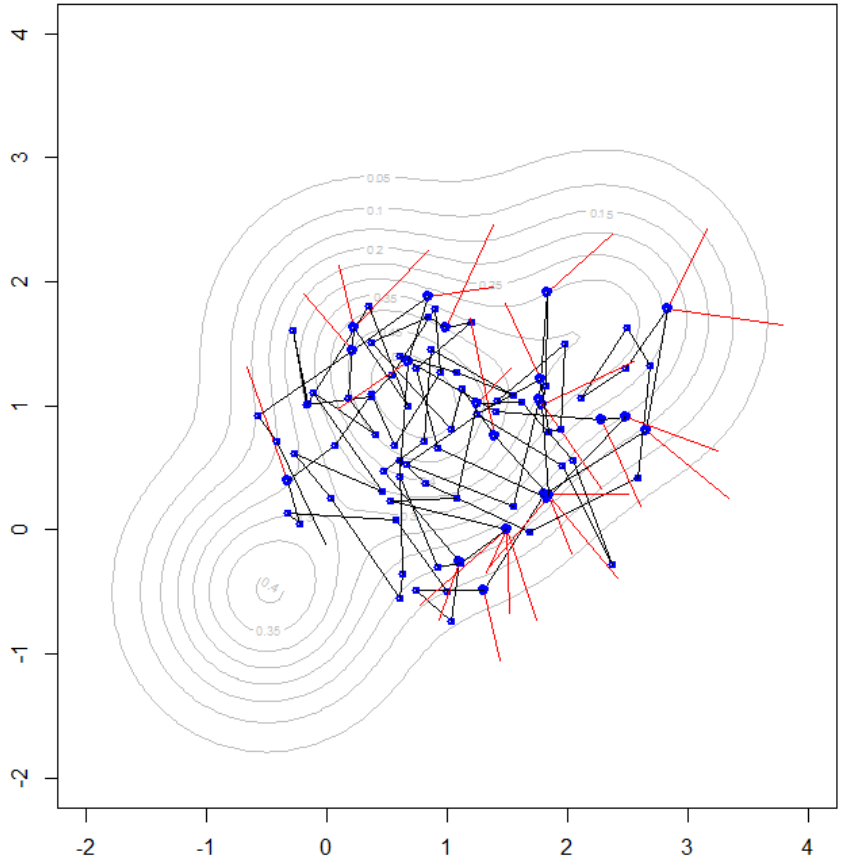
- $f(\boldsymbol{x}^*) = 0.296 > f(\boldsymbol{x}^{(0)}) = 0.098$; so this was an uphill step and is automatically accepted ($R(\boldsymbol{x}^{(t)}, \boldsymbol{x}^*) = \frac{f(\boldsymbol{x}^*)}{f(\boldsymbol{x}^{(t)})} > 1$)

Metropolis alg. – Ex.7



- $\boldsymbol{x}^{(0)}$=(1,-0.5)
- Uphill steps: $\boldsymbol{x}^{(1)}$=(0.58,0.08)
- $\boldsymbol{x}^{(2)}$=(-0.33,0.13)
- $\boldsymbol{x}^{(3)}$=(-0.23,0.05)
- Then downhill step proposed: $\boldsymbol{x}^{*}$=(-0.32,0.4), $R\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{*}\right) = \frac{f(\boldsymbol{x}^{*})}{f(\boldsymbol{x}^{(t)})} = 0.774$
- Random Unif(0,1) generated: 0.573 and since this is smaller than R=0.774, $\boldsymbol{x}^{(4)} = \boldsymbol{x}^{*}$ =(-0.32,0.4) is accepted
- Again downhill step proposed: $\boldsymbol{x}^{*}$=(-0.67,1.31), $R\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{*}\right) = \frac{f(\boldsymbol{x}^{*})}{f(\boldsymbol{x}^{(t)})} = 0.560$; random Unif(0,1): 0.890 and rejection of $\boldsymbol{x}^{*}$
- $\boldsymbol{x}^{(5)} = \boldsymbol{x}^{(4)}$ =(-0.32,0.4)

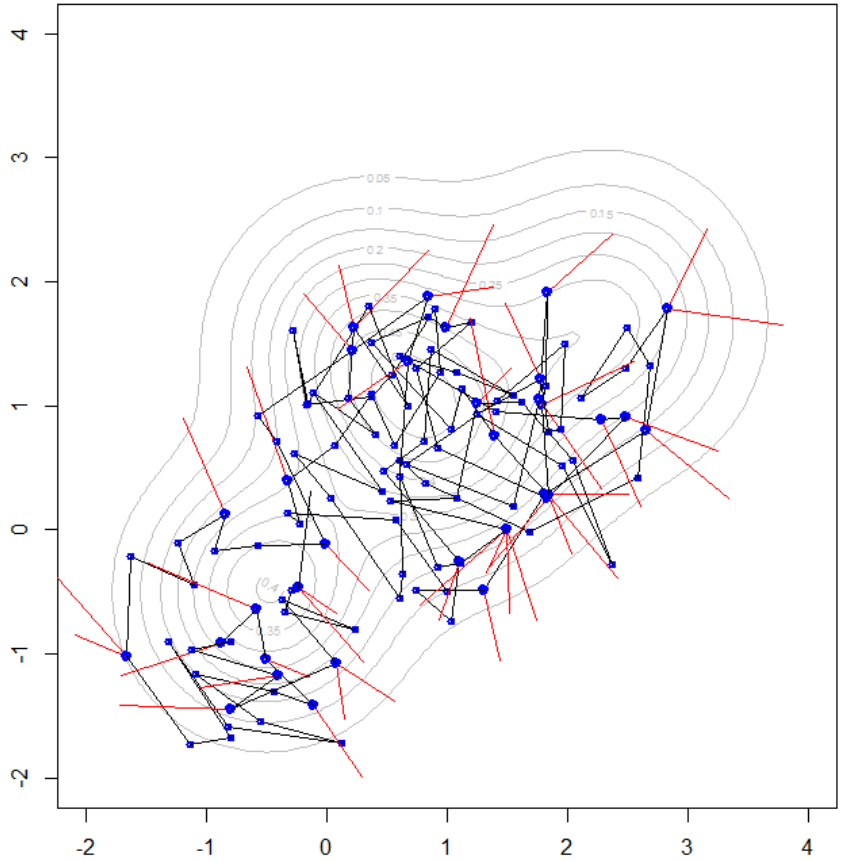
Metropolis alg. – Ex.7



- After several additional iterations (see red lines for rejected proposals), one part of the distribution was explored to a good extend
- Since uphill steps preferred, part of distribution with local maximum at (-0.5,-0.5) is not yet ”detected” at all
- Occasionally, the path will arrive at this part as well

Metropolis alg. – Ex.7

- Now, larger parts of distribution explored



- A couple of animations can be found on: https://chi-feng.github.io/mcmc-demo/app.html#RandomWalkMH,standard (choose Algorithm: RandomWalkMH)

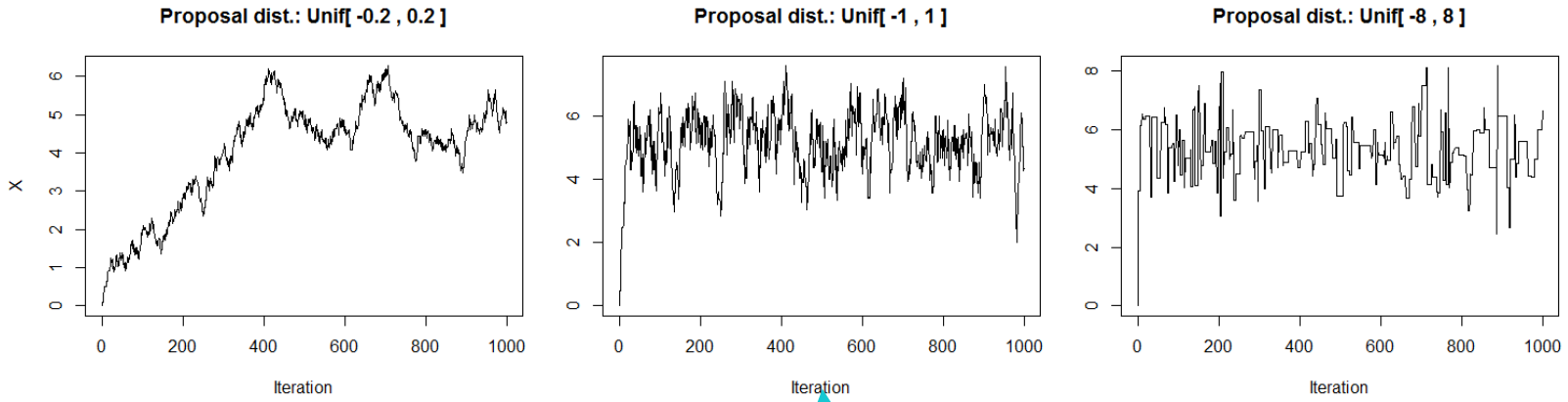
Metropolis algorithm - Example 8

(compare Givens and Hoeting, ex. 5.3)

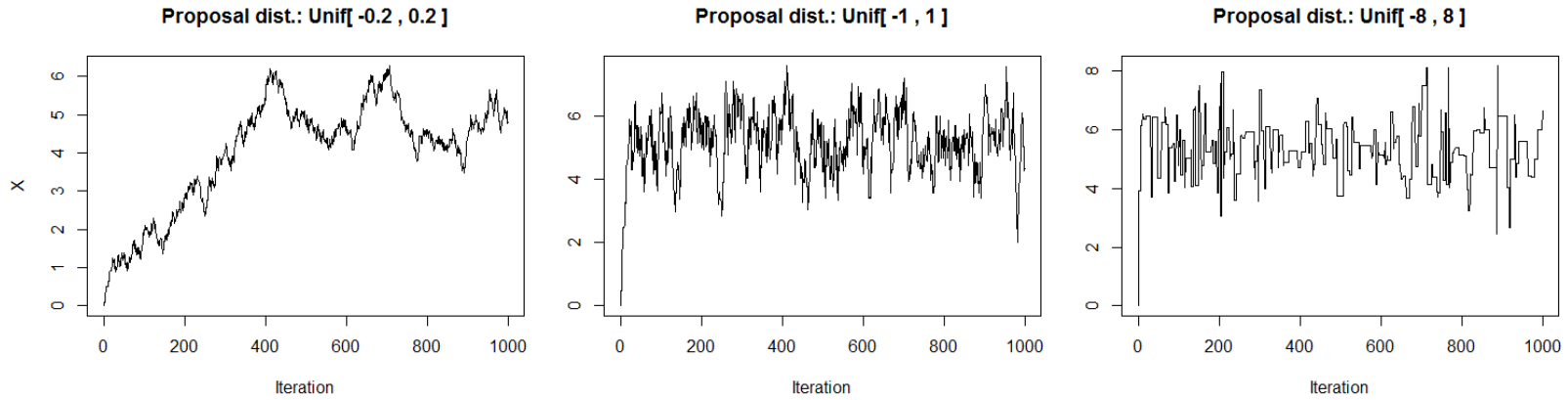
- Consider Bayesian estimation of μ based on $N(\mu, 3^2/7)$ likelihood for μ and Cauchy(5,2) prior; observed mean=5.38
- The posterior density is proportional to product of likelihood and prior density
- **Use MCMC to generate random samples following the posterior density**
- Based on these random samples, one can e.g.
  - determine posterior probability that $2 \leq \mu \leq 8$
  - determine mean and variance of posterior

Metropolis algorithm - Example 8

- We use starting value $x^{(0)} = 0, s = 1000$ iterations and following proposal distributions $g(\cdot\,|x^{(t)})$:
  $x^{(t)}$+Unif[-0.2,0.2], $x^{(t)}$+Unif[-1,1], $x^{(t)}$+Unif[-8,8]
- **Sample path plots** show simulated values $x^{(t)}$ vs. iteration number t



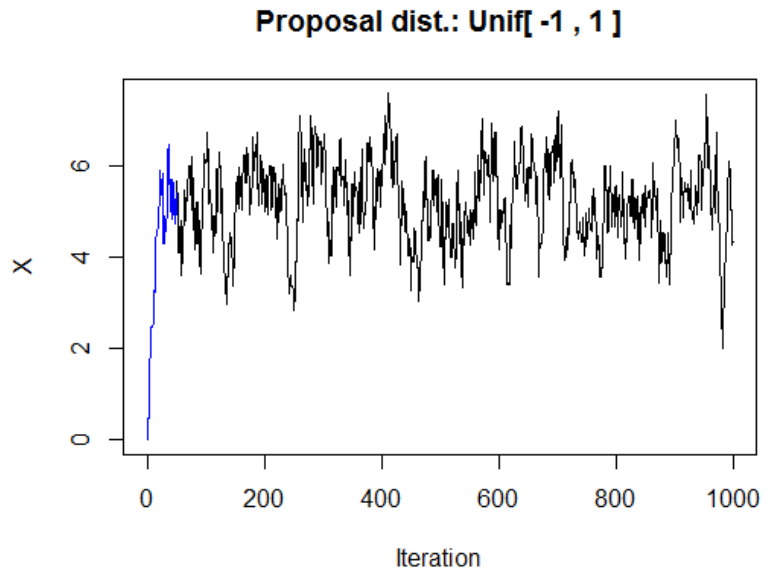
Metropolis algorithm - Example 8

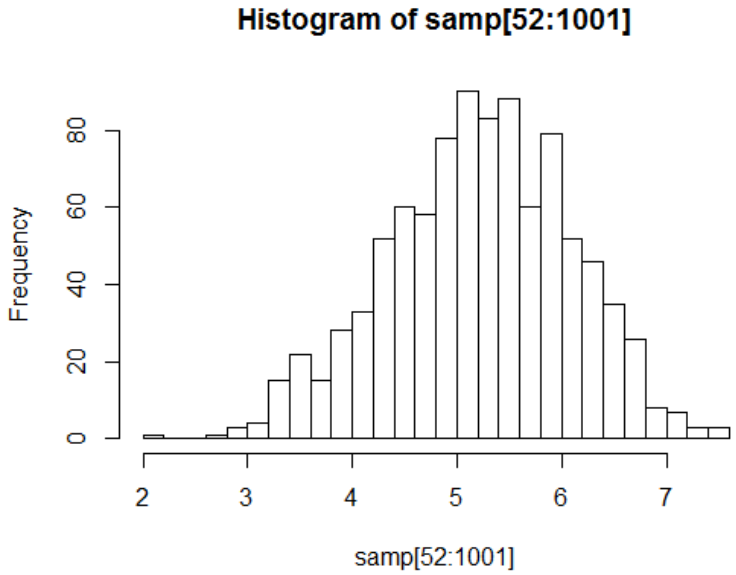


- Count “acceptance rate” (=proportion accepted proposals)
- Here: 98% 78% 18%
- Best results for 44% (uni-dim. case) to 23.4% (high dim. case) acceptance probability (theory based on normal target and proposal functions, see Givens and Hoeting, Chapter 7.3, for references about that)
- For multimodal functions lower acceptance probabilities might be good

Metropolis algorithm - Example 8

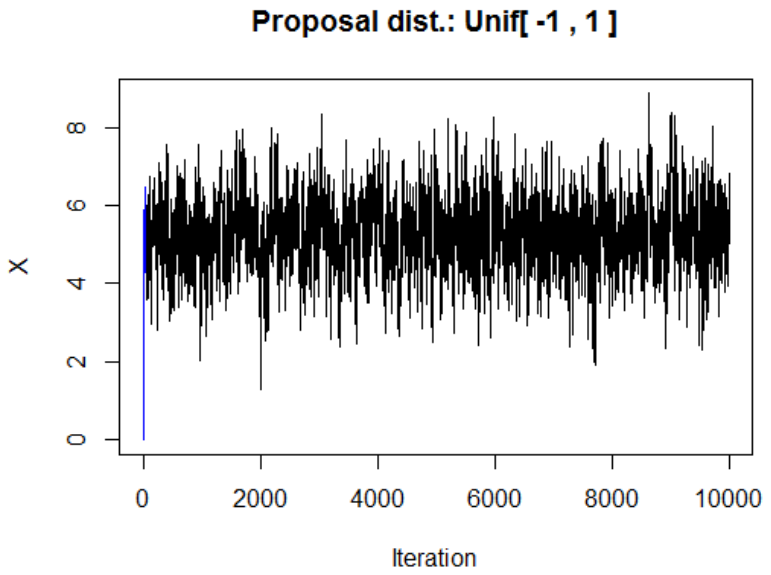
- Based on sample path plots, we might choose $x^{(t)}$+Unif[-1,1] as proposal distribution
- Often, one wants to discard initial samples (**burn-in** period) which highly depend on starting value, e.g., 50 values + $x^{(0)}$

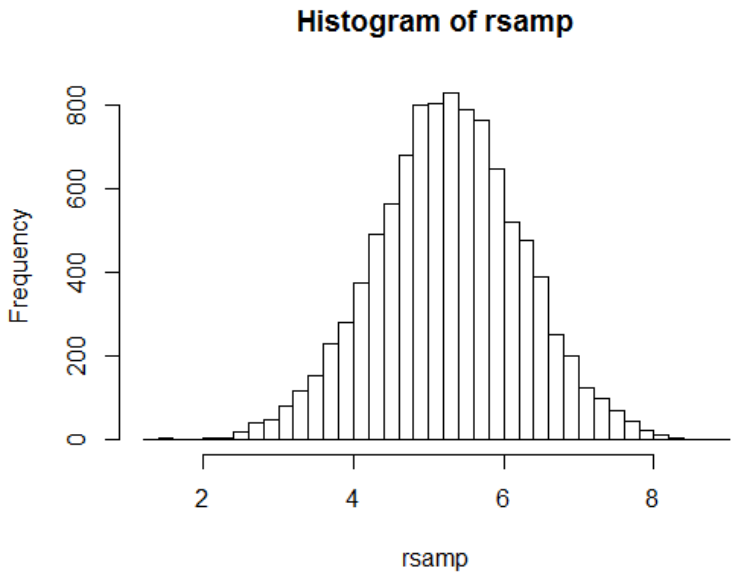




Metropolis algorithm - Example 8

- For $s = 10\,000$ iterations and burn-in of 50, we obtain

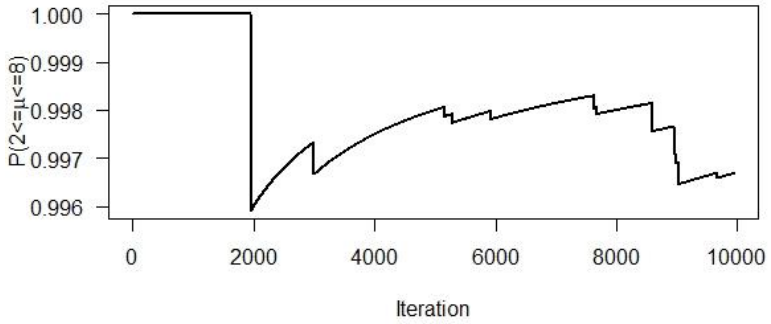




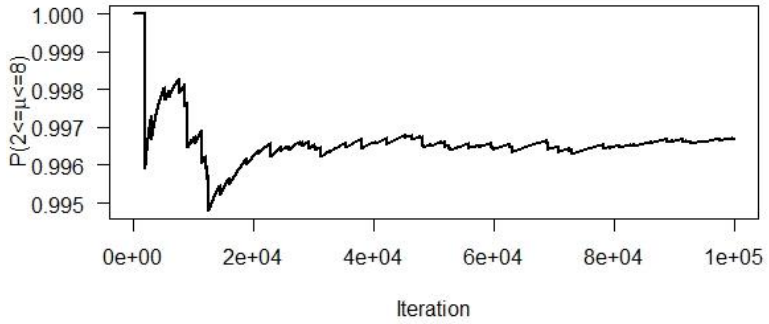
- Monte Carlo estimate for $P(2 \leq \mu \leq 8)$ is 0.9967
  (Monte Carlo standard error=$\sqrt{0.9967 * 0.0033/9950} = 0.0006$)
- Estimated mean = 5.26, standarddeviation =0.99

Metropolis algorithm - Example 8

- Were $s = 10\,000$ iterations enough to ensure convergence to target distribution?
- Can depend on the purpose …
- E.g., for estimating $P(2 \leq \mu \leq 8)$
- One can monitor cusum/convergence plots showing estimate versus iterations (see Givens and Hoeting, ch.7.3.1.1)
- After 10 000 iterations



After 100 000 iterations



- After 10 000 iterations, we might not be happy with the left graph; we run longer and are happy with 100 000

Metropolis-Hastings with independent proposals

- Other proposal distributions g possible (not necessarily symmetric), e.g. independent proposals
- Proposal distribution depends not on previous value, $g(\cdot\,|\boldsymbol{x}^{(t)}) = g(\cdot)$
- The MH ratio is $R\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{*}\right) = \frac{f(\boldsymbol{x}^{*})\, g\left(\boldsymbol{x}^{(t)}\middle|\boldsymbol{x}^{*}\right)}{f\left(\boldsymbol{x}^{(t)}\right) g\left(\boldsymbol{x}^{*}\middle|\boldsymbol{x}^{(t)}\right)} = \frac{f(\boldsymbol{x}^{*})/g(\boldsymbol{x}^{*})}{f\left(\boldsymbol{x}^{(t)}\right)/g(\boldsymbol{x}^{(t)})}$

- A possible application is for Bayesian analysis (f is the posterior) with proposal distribution g being the prior distribution
- f/g is then the likelihood

Markov chain Monte Carlo

- In Givens and Hoeting (2013), Chapter 7 and 8, more about Markov chain Monte Carlo algorithms