UiO : Matematisk institutt
Det matematisk-naturvitenskapelige fakultet

STK-4051/9051 Computational Statistics Spring 2022
Chapter 6 and Note on SMC from Geir Storvik

Instructor: Odd Kolbjørnsen, oddkol@math.uio.no

Integration

- 1D methods for integration $O(n^{-r})$
- Monte Carlo method in higher dimensions (R^d)
  - MC: $O\left(n^{-1/2}\right)$ vs Fubini $O\left(n^{-r/d}\right)$
  - Provided: $\mathrm{var}(h(X)) < \infty$
- Random number generator (RNG)
  - Reproducible randomness = assign seed in a PRNG
- Sampling by inversion and transformation
- Common set up in many cases.
  - Want to sample from $f(x)$, but get sample from $g(x)$
- Rejection sampling: Correct the error by an acceptance step
  - Need a bound of $f(x)/g(x) \leq 1/\alpha$
  - Each sample is a random sample from $f(x)$
  - Need to sample many times to get one sample
    Probability of acceptance α
    Expected number of samples to get acceptance α^{-1}

Importance sampling

- Rewriting

$$\mu = \int h(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} = \int \frac{h(\mathbf{x}) f(\mathbf{x})}{g(\mathbf{x})} g(\mathbf{x}) d\mathbf{x} = \frac{\int \frac{h(\mathbf{x}) f(\mathbf{x})}{g(\mathbf{x})} g(\mathbf{x}) d\mathbf{x}}{\int \frac{f(\mathbf{x})}{g(\mathbf{x})} g(\mathbf{x}) d\mathbf{x}}$$

- Assume $X_1, \ldots, X_n$ iid from $g(\mathbf{x})$. (We know how to sample from g(**x**))

- Two alternative estimates

$$\hat{\mu}^*_{IS} = \frac{1}{n} \sum_{i=1}^{n} h(\mathbf{X}_i) w^*(\mathbf{X}_i), \quad w^*(X_i) = \frac{f(\mathbf{X}_i)}{g(\mathbf{X}_i)}$$

$$\hat{\mu}_{IS} = \sum_{i=1}^{n} h(\mathbf{X}_i) w(\mathbf{X}_i), \quad w(\mathbf{X}_i) = \frac{w^*(\mathbf{X}_i)}{\sum_{j=1}^{n} w^*(\mathbf{X}_j)}$$

- $w^*(\mathbf{X}_i)$ called importance weights
- $w(\mathbf{X}_i)$ called the normalized importance weights

Properly weighted sample

- A weighted random pair (X, W) is properly weighted with respect to π if for any (square integrable) function h

$$E[Wh(X)] = c \cdot E_\pi[h(X)]$$

for some constant c.

- A weighted random sample $\{(X^i, W^i), i = 1, \ldots, N\}$ is properly weighted with respect to π if each (X_i, W_i) are properly weighted.

- Consequence: If $\{(X^i, W^i), i = 1, \ldots, N\}$ are properly weighted iid random pairs, then

$$\hat{\mu} = \frac{\sum_{i=1}^N W^i h(X^i)}{\sum_{i=1}^N W^i} \tag{1}$$

is a consistent estimator of $\mu = E_\pi[h(X)]$ (with respect to increasing N).

Effective sample size

- Assume $w_i = w(\mathbf{X}_i)$, $i = 1, \ldots, n$ are normalized weights
- Define effective sample size by

$$\widehat{N}_{eff} = \frac{1}{\sum_{i=1}^{n} w_i^2}$$

Ex 1:	if $w_i = \frac{1}{n}$ for all i	$\widehat{N}_{eff} = n$
Ex 2:	if $w_i = 0, i \leq z, w_i = \frac{1}{n-z}, i > z$	$\widehat{N}_{eff} = n - z$
Ex 3:	if $w_i = 0, i \neq j, w_j = 1$	$\widehat{N}_{eff} = 1$

Sampling importance resampling

- Assume now we want to sample from $f(\mathbf{x})$, difficult
- Easy to sample from $g(\mathbf{x})$.
- Sampling importance resampling
  1. Sample $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_m$ iid from g
  2. Calculate standardized importance weights

$$w(\mathbf{Y}_i) = \frac{f(\mathbf{Y}_i)/g(\mathbf{Y}_i)}{\sum_{j=1}^{m} f(\mathbf{Y}_j)/g(\mathbf{Y}_j)}, i = 1, \ldots, m$$

  3. Resample $\mathbf{X}_1, \ldots, \mathbf{X}_n$ from $\{\mathbf{Y}_1, \ldots, \mathbf{Y}_m\}$ with probabilities $w(\mathbf{Y}_1), \ldots, w(\mathbf{Y}_m)$
- Properties: As $m \to \infty$
  - X_i converges in distribution to $f(\mathbf{x})$
  - Correlations between X_i's decreases to zero
- For finite m: Correlation between samples

Sampling importance resampling

- Assume
  - $Y_1, \ldots, Y_m$ iid from g
  - $X_1, \ldots, X_n$ resampled from $\{Y_1, \ldots, Y_m\}$, $w(Y_i) = \frac{f(Y_i)}{g(Y_i)}$
- Two possible estimates of $\mu = E^f[X]$:

$$\hat{\mu}_{SIR} = \frac{1}{m}\sum_{i=1}^{m} X_i$$

$$\hat{\mu}_{IS} = \sum_{i=1}^{m} w(Y_i)\, Y_i$$

Can show

$$E[(\hat{\mu}_{IS} - \mu)^2] \leq E[(\hat{\mu}_{SIR} - \mu)^2]$$

Why consider SIR?

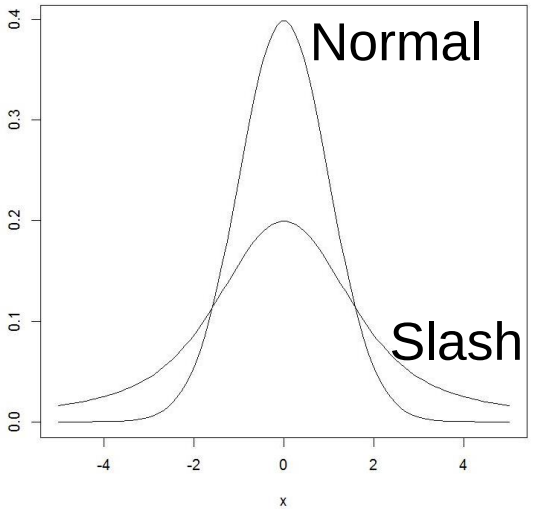
- Sometimes beneficial to have **equally weighted** samples
- May be beneficial at a later stage of analysis process
- If we want to evaluate E(*h(x)*) where *h(x)* is hard to evaluate
- Usually n < m

Example: slash distribution

- Controlled example (we know the truth)
- Y has slash distribution when $Y = \frac{X}{U}$
  X$\sim N(0,1), U \sim \text{Unif}(0,1)$

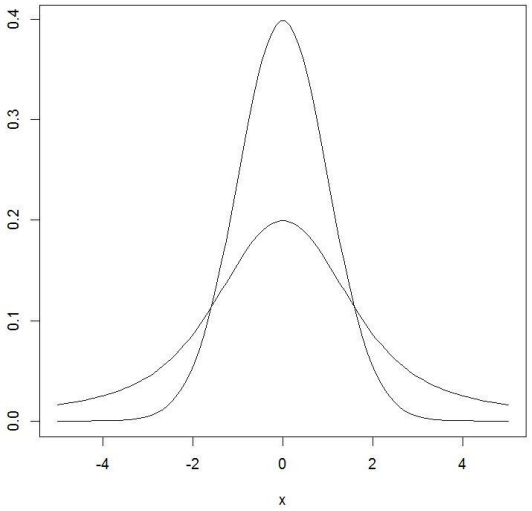
$$f(y) = \begin{cases} \frac{1-\exp\{-y^2/2\}}{y^2\sqrt{2\pi}}, & y \neq 0, \\ \frac{1}{2\sqrt{2\pi}}, & y = 0. \end{cases}$$

- Sampling Experiments
  1. X from Y
  2. Y from X
- Methods
  1. Rejection sampling
  2. Importance sampling
  3. Sampling importance resampling



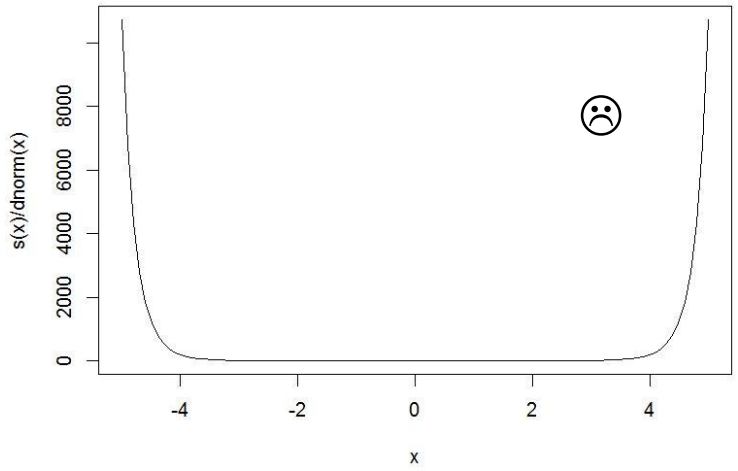
Two test functions

- Ex1: x
- Ex2: h(x)= sin(x)+0.2cos(2*pi*x)

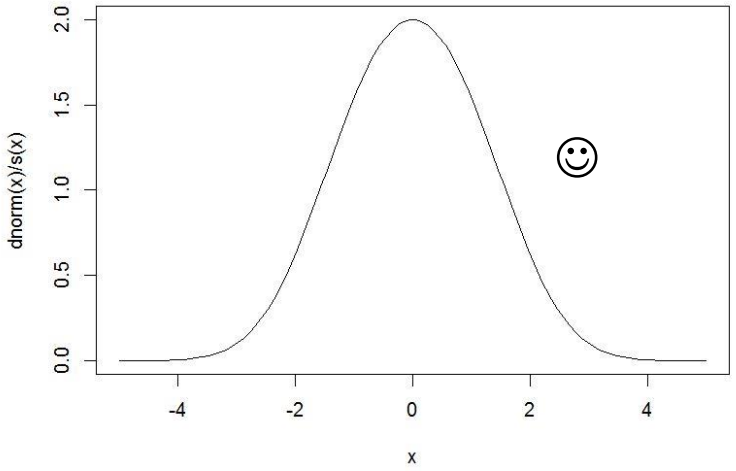


- Ratios:

Slash /normal



Normal/slash



- Ex1: x
- Ex2: h(x)= sin(x)+0.2cos(2*pi*x)

```
> m = 1000
> x=rnorm(m)
> y =rnorm(m)/runif(m)
> show(c(mean(x), mean(h(x)), mean(y), mean(h(y))))
[1] -0.04743281 -0.02314847 -0.71528509  0.01710263
>
>
>
> m = 100000
> x=rnorm(m)
> y =rnorm(m)/runif(m)
> show(c(mean(x), mean(h(x)), mean(y), mean(h(y))))
[1]  0.001369078  0.001019647  0.542154961 -0.004230678
>
>
>
> m = 10000000
> x=rnorm(m)
> y =rnorm(m)/runif(m)
> show(c(mean(x), mean(h(x)), mean(y), mean(h(y))))
[1]  3.115531e-05  4.417861e-05 -8.361942e-01 -2.532640e-05
```

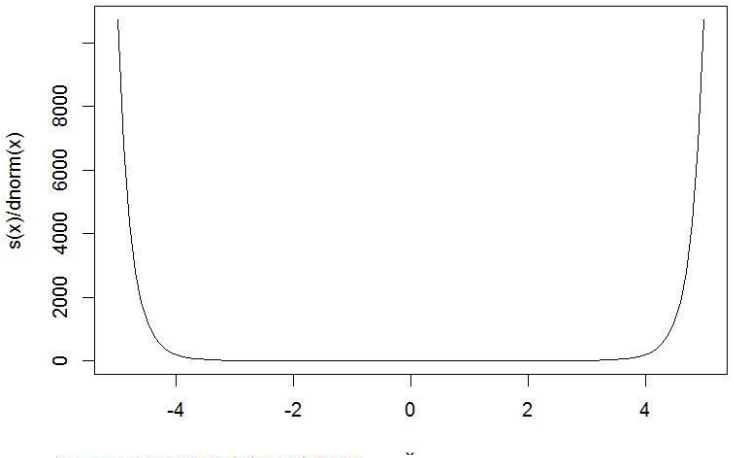
Slash distribution does not have a mean
=> The average does not converge

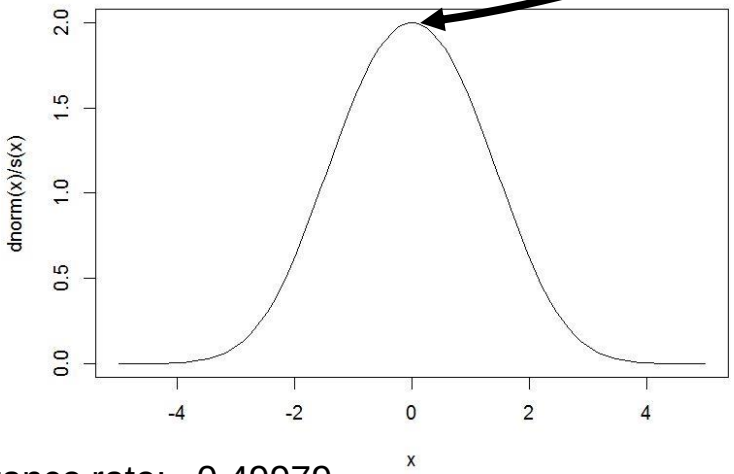
```
> m = 1000
> x=rnorm(m)
> y =rnorm(m)/runif(m)
> show(c(sd(x), sd(h(x)), sd(y), sd(h(y))))
[1]  0.9951861  0.6712401 65.3199546  0.7001361
>
>
> m = 100000
> x=rnorm(m)
> y =rnorm(m)/runif(m)
> show(c(sd(x), sd(h(x)), sd(y), sd(h(y))))
[1]    0.9956829    0.6726304 1328.0501683    0.7122169
>
>
>
> m = 10000000
> x=rnorm(m)
> y =rnorm(m)/runif(m)
> show(c(sd(x), sd(h(x)), sd(y), sd(h(y))))
[1] 9.997296e-01 6.724661e-01 1.110357e+04 7.138005e-01
>
```

Slash distribution does not have a variance
=> The sd(y) increase with sample size

Rejection sampling

- Normal from slash bounded by 2
- Slash from Normal unbounded
  (no rejection sampling possibel)



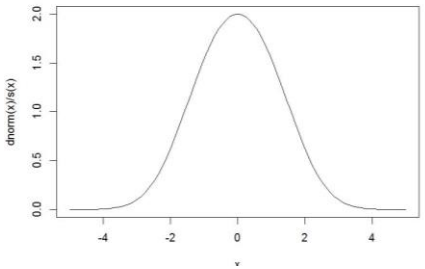


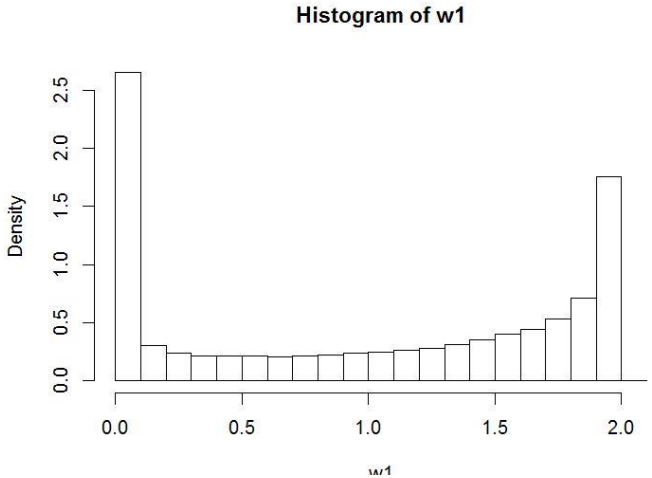

```
> y =rnorm(m)/runif(m)
> U=runif(m)
> accept = dnorm(y)/(s(y)*2)
> sample = y[U<accept]
>
> length(sample)
[1] 49979
> max(accept)
[1] 1
> min(accept)
[1] 0
```

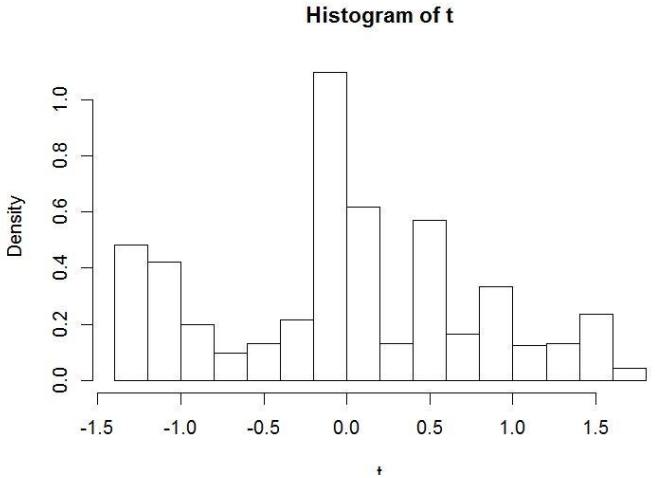
Observed acceptance rate: 0.49979
Theoretical acceptance rate: 0.50000

Sample from slash, estimate propeties of normal distribution

$f(x)/g(x)$



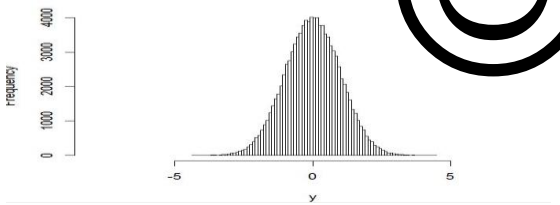





```
> m=100000
> ## importance sampling
> y =rnorm(m)/runif(m)
> w1 = dnorm(y)/s(y)
> wn1 = w1/sum(w1)
> mean(w1)
[1] 1.002287
>
>
> neff = 1/sum(wn1^2)
> show(neff/m)
[1] 0.6213973
```

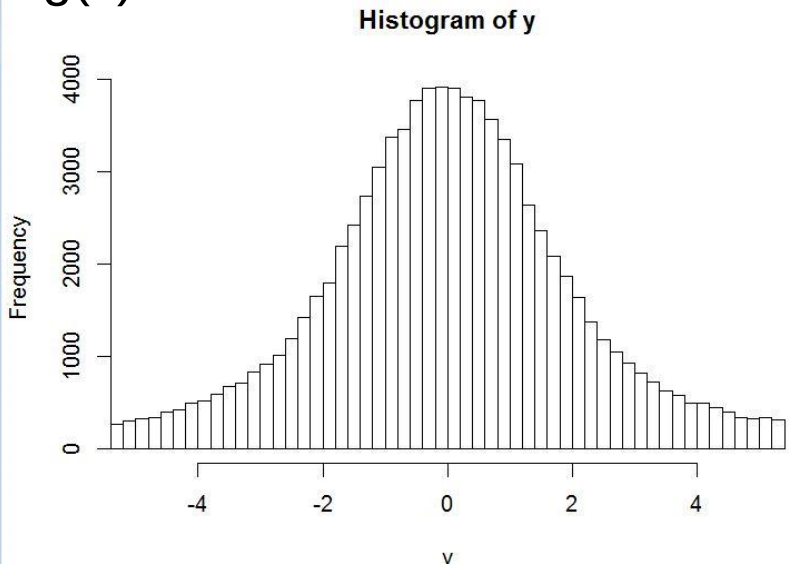
```
>
> t=h(y)*w1
> mean(t)
[1] -0.0007471106
```

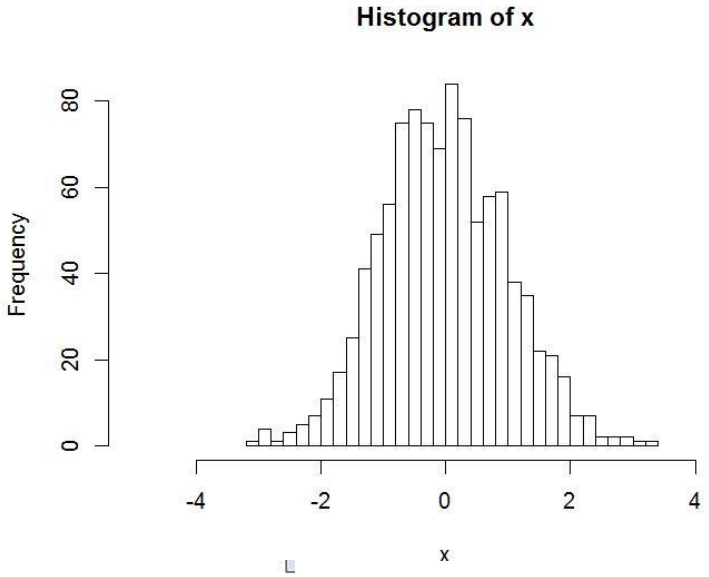
SIR normal from slash



Sample from g(x)

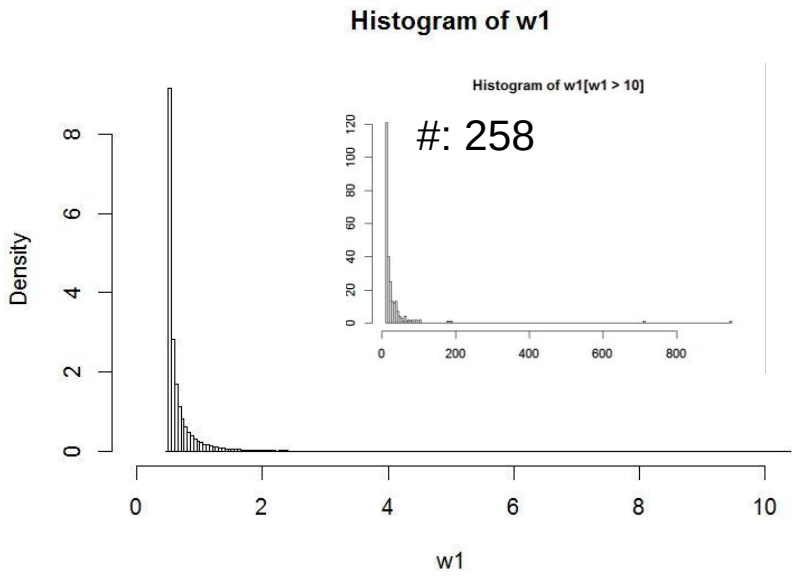
Approximate sample from f(x)

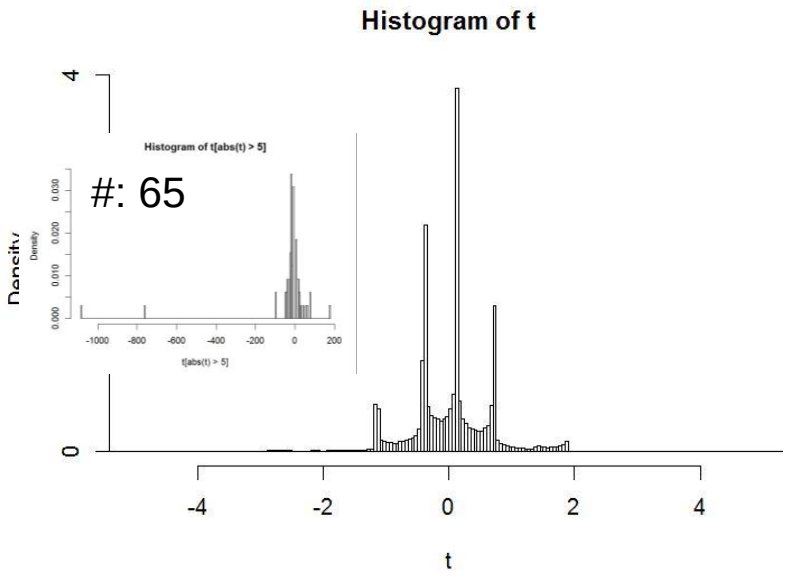





```
n = 1000
x = sample(y,n,replace=T,prob=wn1)
par(mfrow=c(1,2))
hist(y,1000000,xlim=c(-5 ,5))
hist(x,40,xlim=c(-5 ,5))
```

Sample from normal, estimate in slash

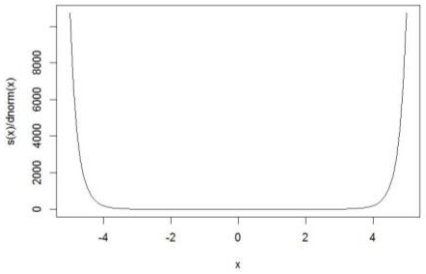


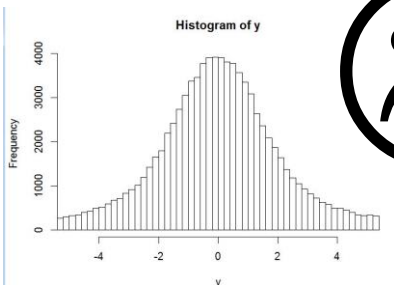



```
> m=100000
> ## importance sampling
> y =rnorm(m)
> w1 = s(y)/dnorm(y)
> wn1 = w1/sum(w1)
> mean(w1)
[1] 0.8170563
>
>
> neff = 1/sum(wn1^2)
> show(neff/m)
[1] 0.03708686
```

Some weigths are very large!
Gives low «effective number samples»

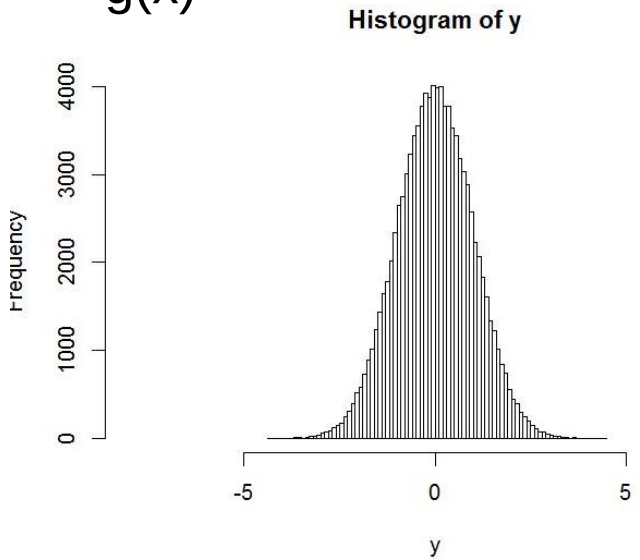
$f(x)/g(x)$



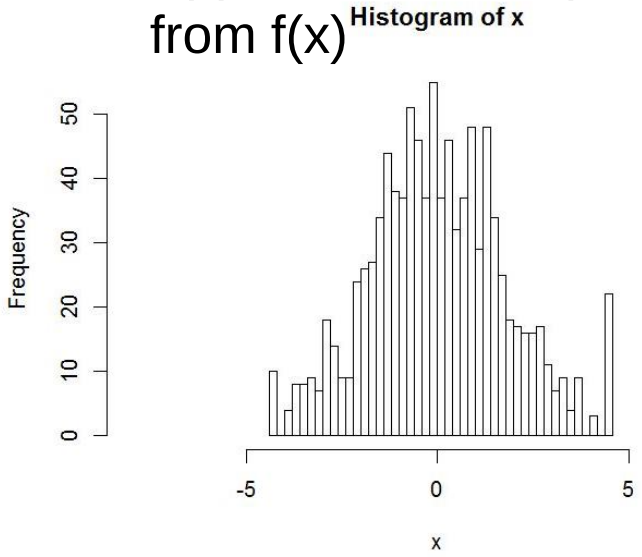


SIR slash from normal

Sample from g(x)

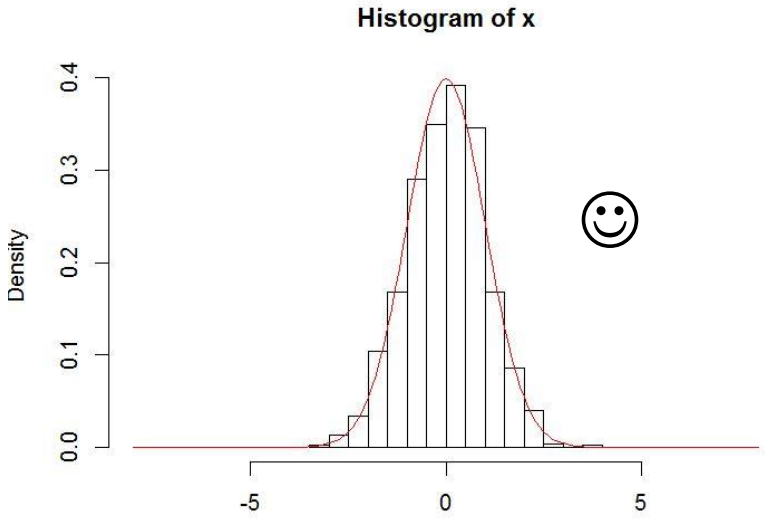


Approximate sample from f(x)

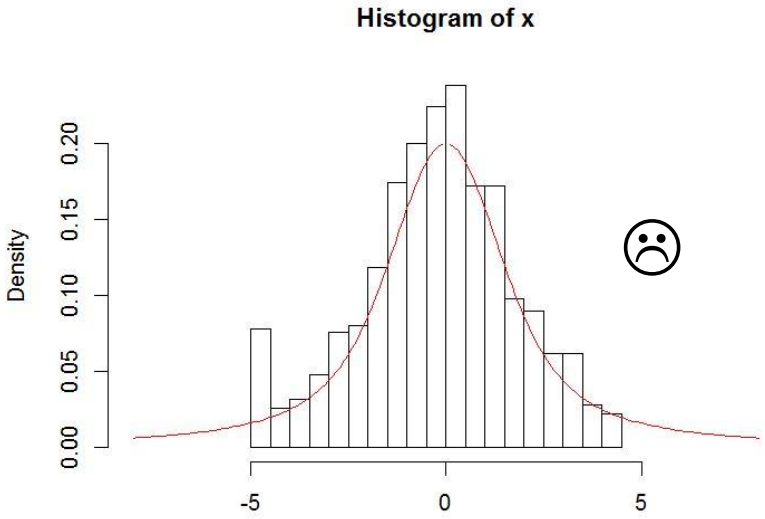



```
> ## SIR resample
> y =rnorm(m)
> w1 = s(y)/dnorm(y)
> wn1 = w1/sum(w1)
> n = 1000
> x = sample(y,n,replace=T,prob=wn1)
> par(mfrow=c(1,2))
> hist(y,1000,xlim=c(-8 ,8))
> hist(x,40,xlim=c(-8 ,8))
```

SIR:
normal from slash and slash from normal




```
> y =rnorm(m)/runif(m)
> w = dnorm(y)/s(y)
> x = sample(y,n,replace=T,prob=w)
> x = sort(x)
> hist(x,20,freq=F,xlim=c(-8,8))
> xp=seq(-8,8,by=0.1)
> lines(xp,dnorm(xp),col=2)
```



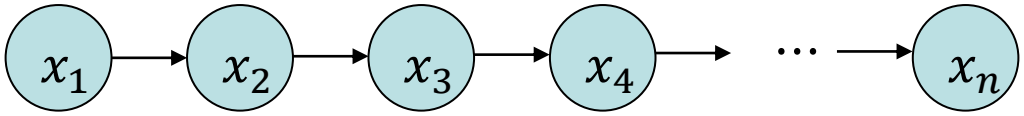

```
> y = rnorm(m)
> w = s(y)/dnorm(y)
> x = sample(y,n,replace=T,prob=w)
> x = sort(x)
> hist(x,20,freq=F,xlim=c(-8,8))
> lines(xp,s(xp),col=2)
```

Problem with presented methods in high dimensions, Rejection sampling

- Assume you want to sample iid $X_i, i = 1, \dots, n$
- Two methods with rejection sampling $\frac{f(x_i)}{g(x_i)} \leq \alpha^{-1}$
  1. Sample $X_i, i = 1, \dots, n$ independently.
  2. Sample $\{X_1, X_2, \dots, X_n\}$ simultaniously
- Expected time to sample complete
  1. $n \cdot \alpha^{-1}$ (nice ☺)
  2. $1 \cdot \alpha^{-n}$ (curse of dimensionality is back 😐)
- For complex distributions in high dimensions we need a joint proposal (within this framework)

Factoring into 1D distributions

- Target
  - $f(\boldsymbol{x}) = f(x_1)f(x_2|x_1)\cdots f(x_n|x_1, x_2, \ldots, x_{n-1})$
- Simpler to sample from
  - $g(\boldsymbol{x}) = g(x_1)g(x_2|x_1)\cdots g(x_n|x_1, x_2, \ldots, x_{n-1})$
- Sample each component sequentially as 1D
- Markov property: (same principle - simpler notation)
  - $f(\boldsymbol{x}) = f(x_1)f(x_2|x_1)f(x_3|x_2)\cdots f(x_n|x_{n-1})$
  - $g(\boldsymbol{x}) = g(x_1)g(x_2|x_1)g(x_3|x_2)\cdots g(x_n|x_{n-1})$



Sequential Monte Carlo in a Markov structure

- Sequential Monte Carlo: First high-dimensional setting
- Assume now $\mathbf{x} = \mathbf{x}_{1:t} = (x_1, \ldots, x_t)$ have a Markov structure

$$f_t(\mathbf{x}_{1:t}) = f_1(x_1) \prod_{i=2}^{t} f_i(x_i|x_{i-1})$$

- Also assume a proposal distribution with Markov property:

$$g_t(\mathbf{x}_{1:t}) = g_1(x_1) \prod_{i=2}^{t} g_i(x_i|x_{i-1})$$

- Importance weights:

$$w(\mathbf{x}_{1:t}) = \frac{f_t(\mathbf{x}_{1:t})}{g_t(\mathbf{x}_{1:t})} = \frac{f_1(x_1)}{g_1(x_1)} \prod_{i=2}^{t} \frac{f_i(x_i|x_{i-1})}{g_i(x_i|x_{i-1)})} = w(\mathbf{x}_{1:t-1}) \frac{f_t(x_t|x_{t-1})}{g_t(x_t|x_{t-1)})}$$

- Opens up for sequential sampling/estimation
- Note: Easy to generalize to non-Markov settings as well
  - More computing at each step

Sequential Monte Carlo

Algorithm

1. Sample $X_1 \sim g_1(\cdot)$. Let $w_1 = u_1 = f_1(x_1)/g_1(x_1)$. Set $t = 2$
2. Sample $X_t|x_{t-1} \sim g_t(x_t|x_{t-1})$.
3. Append x_t to $\mathbf{x}_{1:t-1}$, obtaining $\mathbf{x}_t$
4. Let $u_t = f_t(x_t|x_{t-1})/g_t(x_t|x_{t-1})$
5. Let $w_t = w_{t-1}u_t$, the importance weight for $\mathbf{x}_{1:t}$
6. Increment t and return to step 2

Can simulate m sequences in paralell!

Sequential Monte Carlo Example

- Assume Markov model

$$X_1 \sim N(0, \sqrt{2})$$

$$f_t(x_t|x_{t-1}) \propto |\cos(X_t - X_{t-1})| \exp\left\{-\tfrac{1}{4}(X_t - X_{t-1})^2\right\}$$

- Of interest:

$$\mu_t = E[X_t] \qquad = 0 \text{ due to symmetry}$$

$$\sigma_t^2 = \text{var}(X_t) = E[X_t^2] - (E[X_t])^2$$

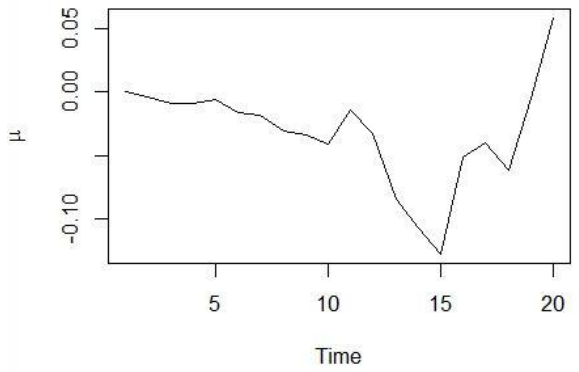
- Estimators by SMC simulation:

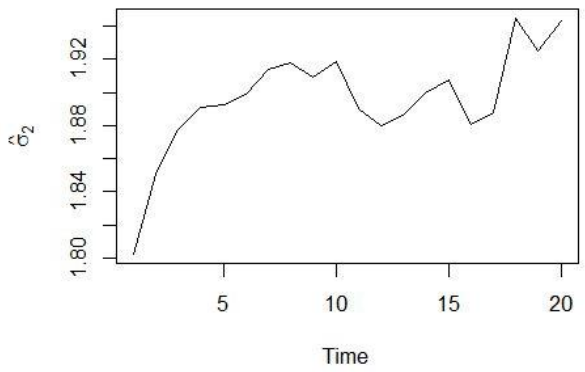
$$\hat{\mu}_t = \frac{\sum_{i=1}^n w_t^i x_t^i}{\sum_{i=1}^n w_t^i}$$

$$\hat{\sigma}_t^2 = \frac{\sum_{i=1}^n w_t^i (x_t^i)^2}{\sum_{i=1}^n w_t^i} - \left[\frac{\sum_{i=1}^n w_t^i x_t^i}{\sum_{i=1}^n w_t^i}\right]^2 = \frac{\sum_{i=1}^n w_t^i (x_t^i - \hat{\mu}_t)^2}{\sum_{i=1}^n w_t^i}$$

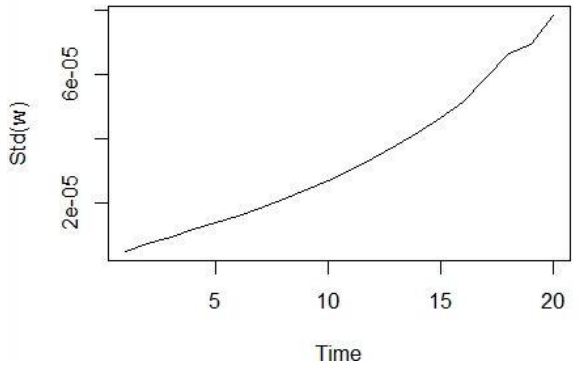
- `SMC_cosnorm.R`

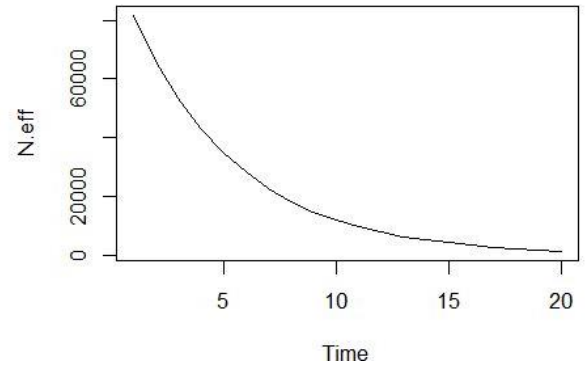
Dimension =20, m=100 000





$$\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{n} w_i^2}$$

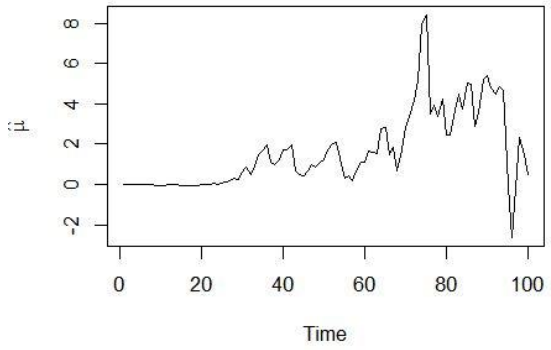


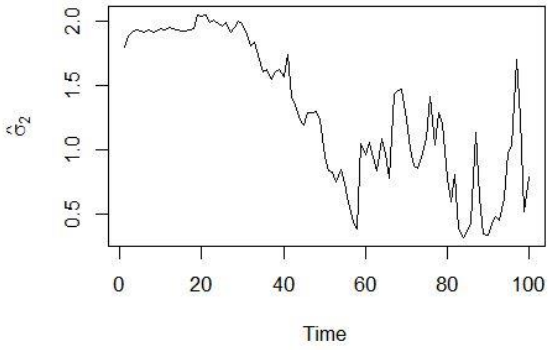


Neff: 1514

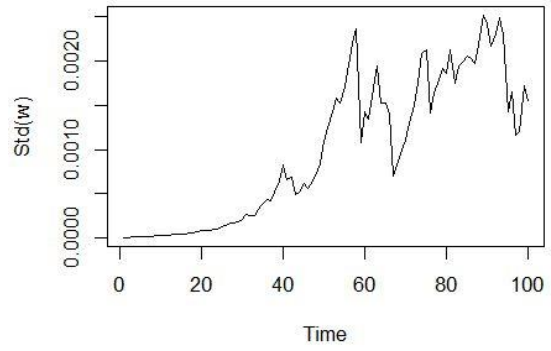
```
show(w.norm.sort[1:10])
[1] 0.007377638 0.006993864 0.004586039 0.003991772 0.003901192 0.003686038 0.003401063 0.003339764 0.003094683 0.002911
```

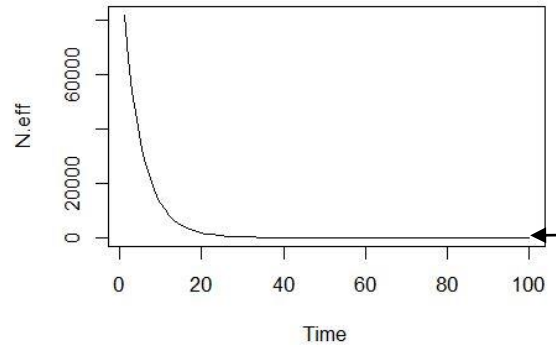
Dimension =100, m=100 000





$$\widehat{N}_{eff} = \frac{1}{\sum_{i=1}^{n} w_i^2}$$





Neff: 3.78

```
show(w.norm.sort[1:10])
[1] 0.47751939 0.06007770 0.04960048 0.03573471 0.03281825 0.02935132 0.02665531 0.02271621 0.02231759 0.02197812
```

Weight degeneracy

- General rule:

$$\text{var}[Y] = E[\text{var}[Y|Z]] + \text{var}[E[Y|Z]] \geq \text{var}[E[Y|Z]]$$

- $Y = w_t, Z = \mathbf{X}_{1:t-1}$ (w_{t-1} given by $\mathbf{x}_{1:t-1}$):

$$\begin{aligned} E[w_t|X_{1:t-1}] =& w_{t-1} E[\tfrac{f_t(X_t|X_{t-1})}{g_t(X_t|X_{t-1})}|\mathbf{X}_{1:t-1}] \\ =& w_{t-1} \cdot 1 = w_{t-1} \end{aligned}$$

  implying that

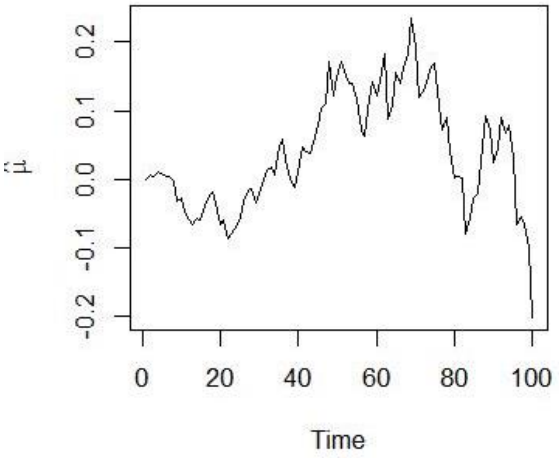
$$\text{var}[w_t] \geq \text{var}[w_{t-1}]$$

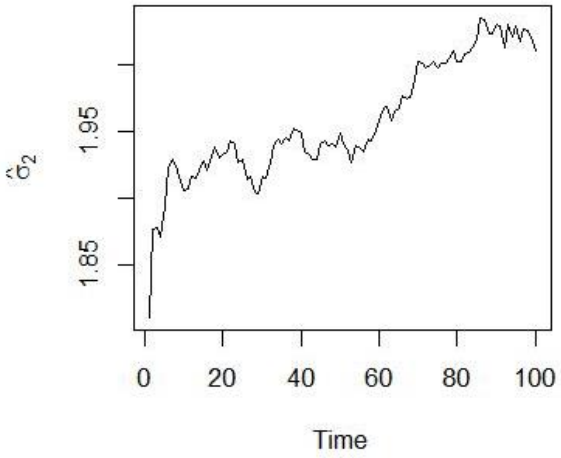
  which indicates that the variance will increase at each time-step.
- Practical consequence:
  - Only a few samples will dominate the others
  - Variability of estimate will increase

Resampling

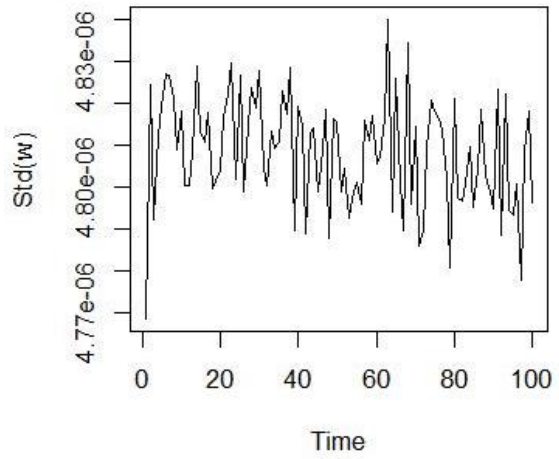
- Degeneracy of weights a serious problem.
- Solution: Resampling (SIR idea)
  - One possible choice for resampling: Apply SIR directly
  - Each iteration:
    - Sample $\tilde{x}_t^i$, independently from $\{x_t^i\}$, probability of each sample is w_t^i
    - Set all new weigths equal to 1/N
  - When N_{eff} is small
    - Sample $\tilde{x}_t^i$, independently from $\{x_t^i\}$, probability of each sample is w_t^i
    - Set all new weigths equal to 1/N
- `SMC_cosnorm.R`

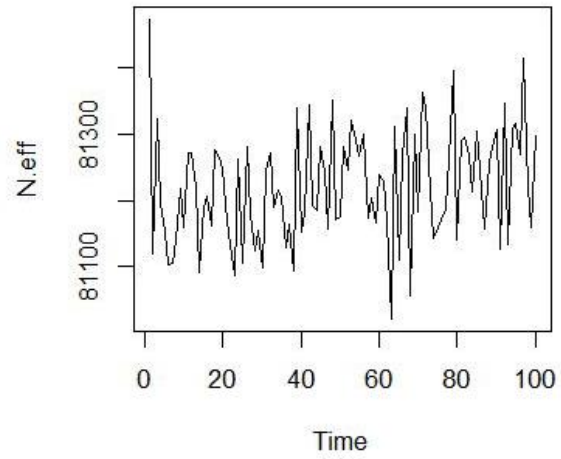
Dimension =100, m=100 000



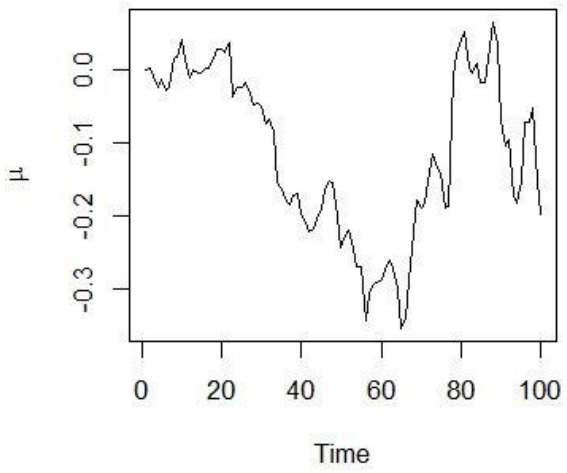


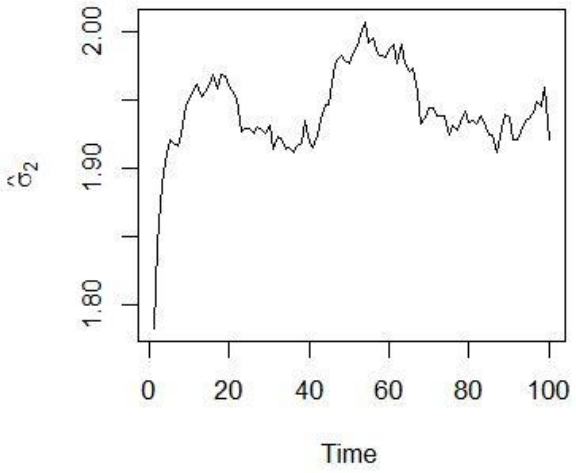
Resample each time



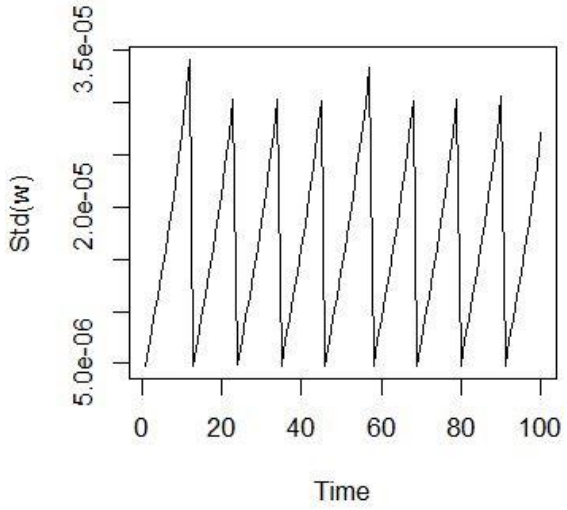


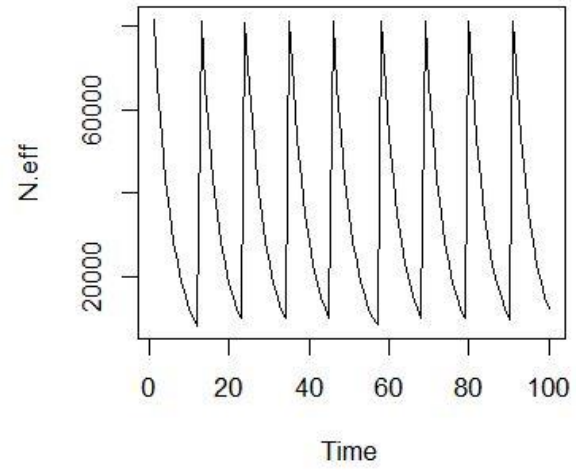
Dimension =100, m=100 000





Resample
n<10 000





How to resample more generally?

- Resampling $\tilde{x}_t^i$ from x_t^i :
  - Original (normalized) weights: w_t^i
  - Resampling weights: $\widetilde{w_t}^i$
  - The number of repeats for x_t^i: N_t^i
- We need the expected number of resamples times the new weight to be the old weight
  - $E\left(N_t^i \cdot \widetilde{w_t}^i\right) = w_t^i$
- One choice (as above)
  - Sample $\tilde{x}_t^i$, independently from $\{x_t^i\}$, probability of each sample is w_t^i
  - Set new weights 1/N

Resampling

Original (normalized) weights: w_t^i

Resampling weights: $\widetilde{w}_t^i$

The number of repeats for x_t^i: N_t^i

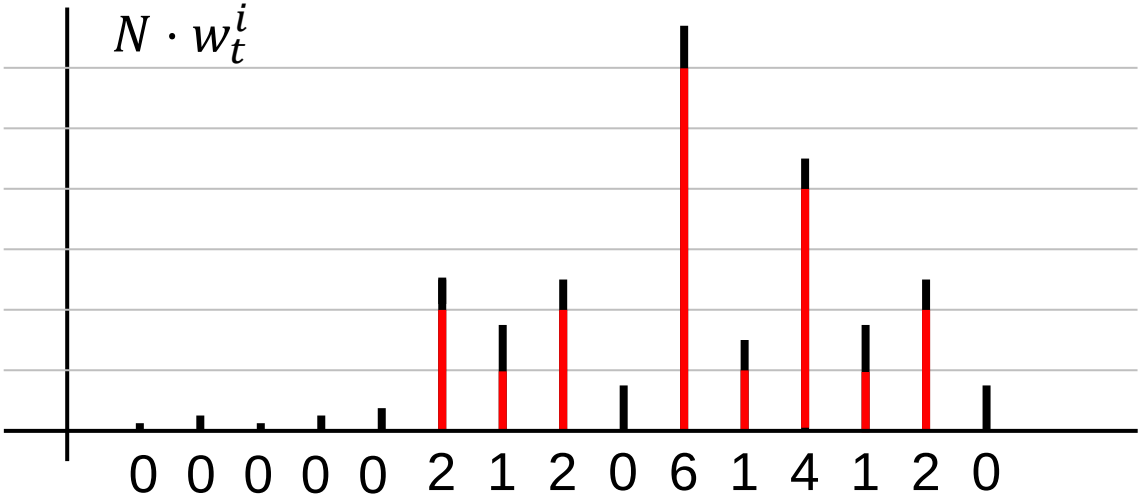
- Simplest option:
  - Resample with probabilities equal to w_t^i.
  - Put weights on resample to $\tilde{w}_t^i = N^{-1}$
  - Number of repeats of x_t^i, N_t^i is Binomial(N, w_t^i)
  - $E[N_t^i \tilde{w}_t^i] = N w_t^i$
- More general resampling strategies are possible
- Sufficient requirement: $E[N_t^i \tilde{w}_t^i] = N w_t^i$
- Optimal strategy (for equally weighted samples)
  - For $i = 1, \ldots, N$, put ($\lfloor a \rfloor$ is the largest integer smaller than a)

$$\widetilde{N}_t^i = \lfloor N w_t^i \rfloor \quad \text{(Some will be zero)}$$

  - Let $\delta_t^i = w_t^i - \widetilde{N}_t^i / N$
  - Define $K = N - \sum_{i=1}^N \widetilde{N}_t^i$ (remaining particles that have not been allocated)
  - Sample $(D_t^1, \ldots, D_t^N)$ from the multinomial distribution with probabilities proportional to $(\delta_t^1, \ldots, \delta_t^n)$.
  - Put $N_t^i = \tilde{N}_t^i + D_t^i$
  - Make N_t^i replicates of x_t^i, but all weights to $1/N$

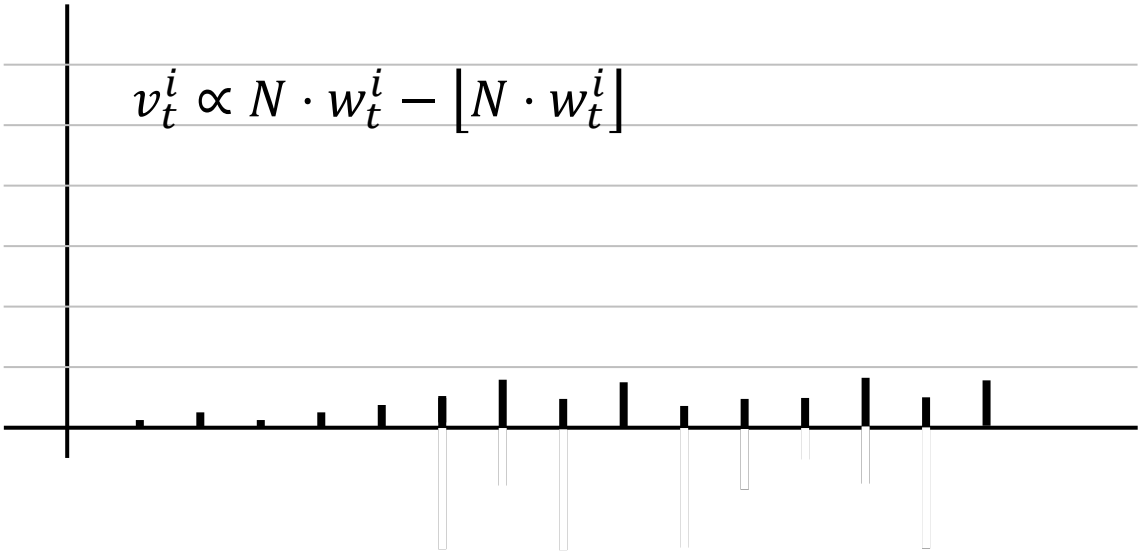
Illustration of optimal resampling

(resample weigth: $1/N$**)**



N=25

Total=19



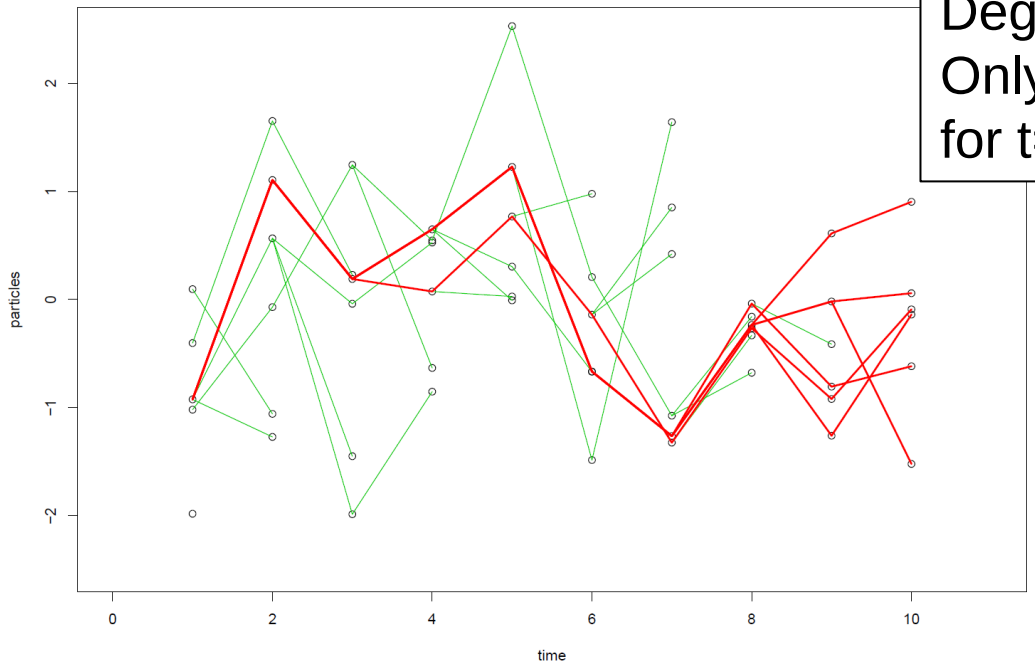
Resample=6

Example optimal resampling

- $N = 5$, $\mathbf{w} = (0.3, 0.4, 0.05, 0.15, 0.2)$
- $N * \mathbf{w} = (1.5, 2.0, 0.25, 0.75, 1.0)$
- $\lfloor N \cdot w_t^i \rfloor$ $\tilde{\mathbf{N}} = (1, 2, 0, 0, 1)$
- $K = 5 - 4 = 1$
- $\tilde{\mathbf{N}}/N = (0.2, 0.4, 0.0, 0.0, 0.2)$
- $\boldsymbol{\delta} = (0.1, 0.0, 0.05, 0.15, 0.0)$
- Sample $\mathbf{D}$ from Multinom$(1 : N, 1, (\frac{0.1}{0.3}, \frac{0.0}{0.3}, \frac{0.05}{0.3}, \frac{0.15}{0.3}, \frac{0.0}{0.3}))$
  e.g $\mathbf{D} = (1, 0, 0, 0, 0)$
- Put $\mathbf{N} = \tilde{\mathbf{N}} + \mathbf{D} = (2, 2, 0, 0, 1)$

Resampling properties

- Resampling will introduce extra random noise at the current time-point
- Can reduce noise at later time points
- Gives a good approximation to $f(x_n)$



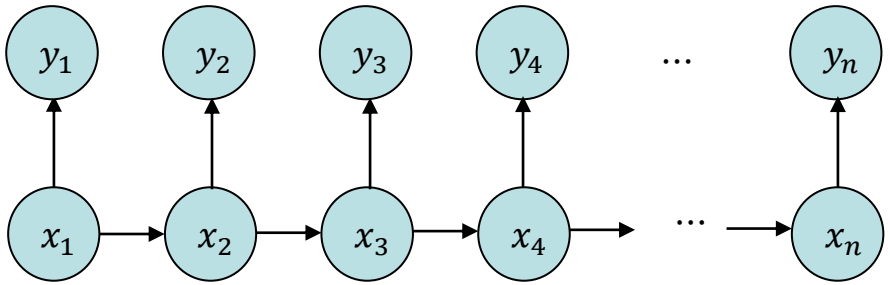
Degenerated sample path
Only one particle left
for t=1,2 and 3

- Does not give a good approximation to $f(\mathbf{x})$ or $f(x_1)$!

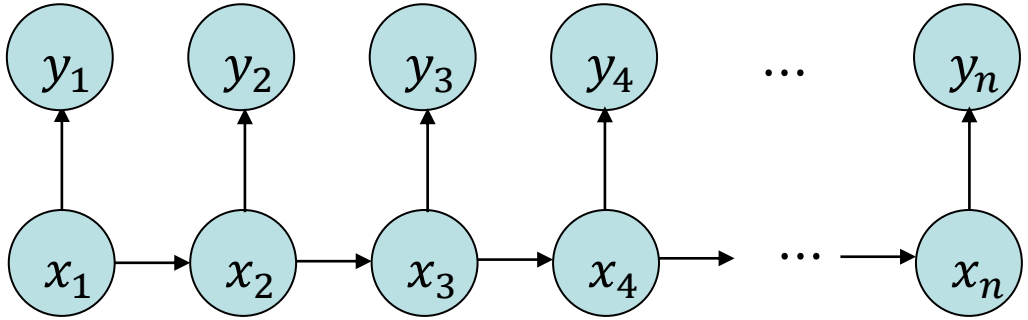
Resampling properties

- Resampling will introduce extra random noise at the current time-point
- Can reduce noise at later time points
- Gives a good approximation to $f(x_n)$
- Does not give a good approximation to $f(\mathbf{x})$ or $f(x_1)$!

Makes it suited for filter problem $p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$

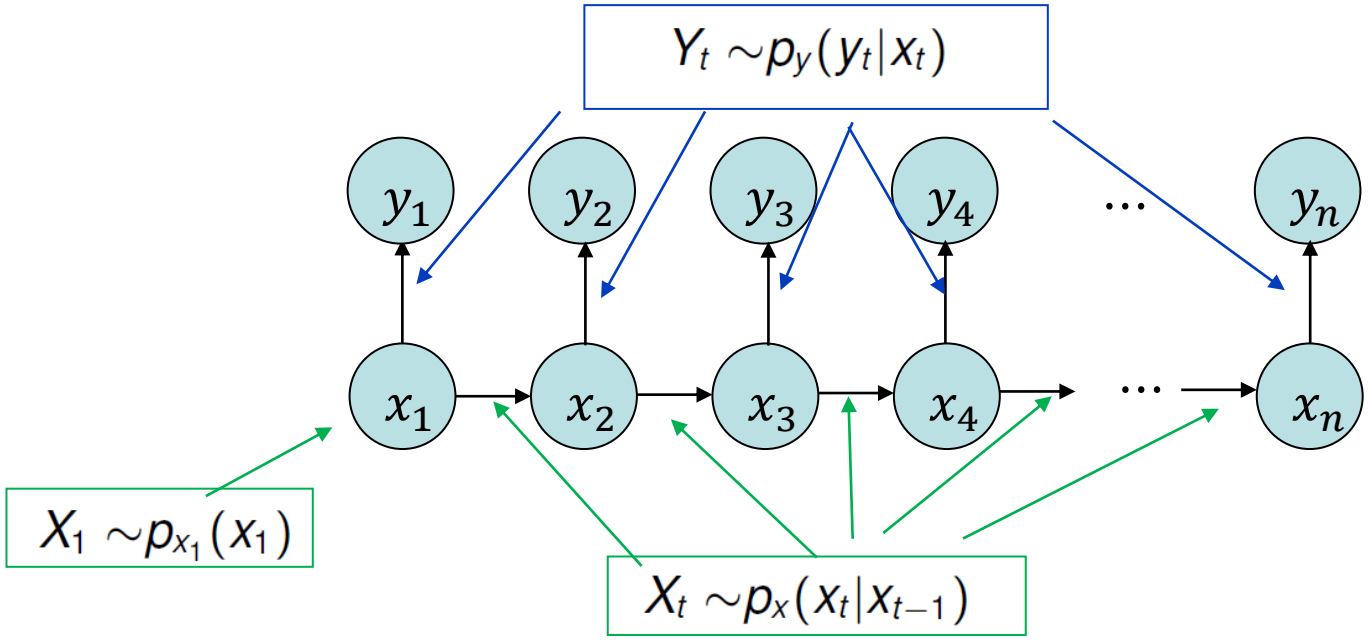


Origin of method in state space models



- For the filtering problem the target is the distribution $p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$
- In chapter 4, we consider a hidden Markov model the distribution of x (state) is discrete here x (the state) it can be continuous

Origin of method in state space models



- For the filtering problem the target is the distribution $p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$
- Target: $f_t(x_t|x_{t-1}) \propto p(x_t|x_{t-1})p(y_t|x_t)$
- Proposal: $g_t(x_t|x_{t-1}) \propto p(x_t|x_{t-1})$

Model for GPS positioning

Model:

$x_0 \sim N(0, \Sigma)$

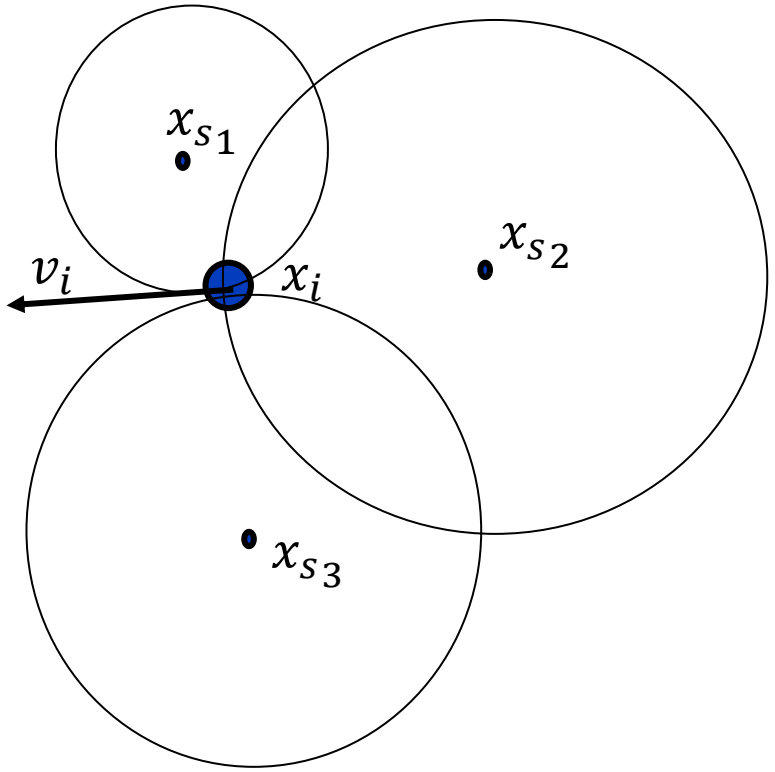
$x_{i+1} = x_i + v_i \cdot \Delta t$

$v_{i+1} = v_i + \Delta v_i + \varepsilon_{v,i}$

$$d_{i,1} = \left\| x_i - x_{s_{2,i}} \right\| + \varepsilon_{d,i,1}$$

$$d_{i,2} = \left\| x_i - x_{s_{3,i}} \right\| + \varepsilon_{d,i,2}$$

$$d_{i,3} = \left\| x_i - x_{s_{2,i}} \right\| + \varepsilon_{d,i,3}$$

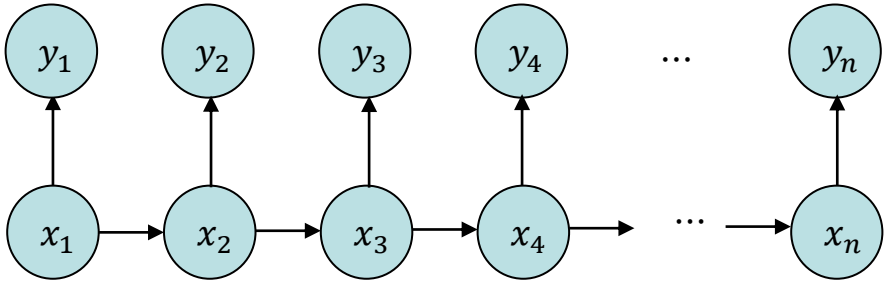


How can this problem be solved using SMC?

Hidden Markov models - state space models

- Assume

$$X_1 \sim p_{x_1}(x_1)$$
$$X_t \sim p_x(x_t|x_{t-1})$$
$$Y_t \sim p_y(y_t|x_t)$$



- $\{y_t\}$ observed, $\{x_t\}$ hidden
- Chapter 4: $\{x_t\}$ discrete. Now possibly continuous
- Aim: $f(\mathbf{x}_{1:t}|\mathbf{y}_{1:t})$ or $f(\mathbf{x}_t|\mathbf{y}_{1:t})$
- Recursive relationship (misprint in book):

$$f(\mathbf{x}_{1:t}|\mathbf{y}_{1:t}) = \frac{f(\mathbf{x}_{1:t}, y_t|\mathbf{y}_{1:t-1})}{f(y_t|\mathbf{y}_{1:t-1})}$$
$$= \frac{f(\mathbf{x}_{1:t-1}|\mathbf{y}_{1:t-1})p_x(x_t|x_{t-1})p_y(y_t|x_t)}{f(y_t|\mathbf{y}_{1:t-1})}$$
$$\propto f(\mathbf{x}_{1:t-1}|y_{1:t-1})p_x(x_t|x_{t-1})p_y(y_t|x_t)$$

Sequential Monte Carlo and HMM

- Assume $g_t(x_t|x_{t-1}) = p_x(x_t|x_{t-1})$

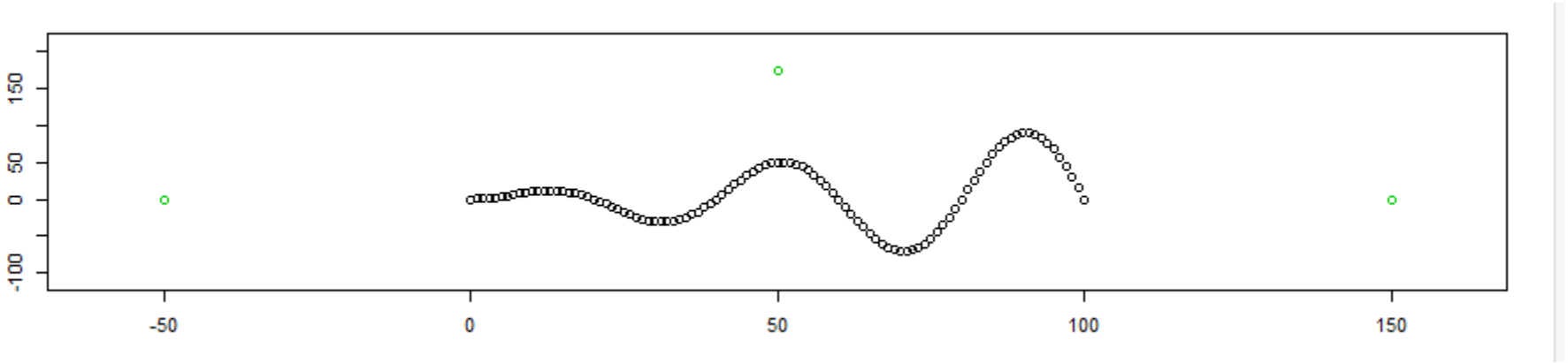
$$
\begin{aligned}
w_t =& \frac{f(\mathbf{x}_{1:t}|\mathbf{y}_{1:t})}{g(\mathbf{x}_{1:t})} \\
\propto& \frac{f(\mathbf{x}_{1:t-1}|y_{1:t-1})p_x(x_t|x_{t-1})p_y(y_t|x_t)}{p_{x_1}(x_1)\prod_{s=2}^{t} p_x(x_s|x_{s-1})} \\
=& \frac{f(\mathbf{x}_{1:t-1}|y_{1:t-1})}{g(\mathbf{x}_{1:t-1})} \frac{p_x(x_t|x_{t-1})p_y(y_t|x_t)}{p_x(x_t|x_{t-1})} \\
=& w_{t-1} p_y(y_t|x_t)
\end{aligned}
$$

- Algorithm
  1. Sample $X_1^i \sim p_{x_1}(\cdot)$, $i = 1, \ldots, n$.
  2. Let $w_1^{*i} = u_1^i = p_y(y_1|x_1^i)$, normalize to $w_i^i = w_1^{*i} / \sum_j w_1^{*j}$. Set $t = 2$
  3. Sample $X_t^i|x_{t-1}^i \sim p_x(x_t|x_{t-1}^i)$, $i = 1, \ldots, n$.
  4. Append x_t^i to $\mathbf{x}_{1:t-1}^i$, obtaining $\mathbf{x}_t^i$
  5. Let $u_t^i = p_y(y_t|x_t^i)$
  6. Let $w_t^{*i} = w_{t-1}^i u_t^i$, normalize to $w_t^i = w_t^{*i} / \sum_j w_t^{*j}$.
  7. If $\hat{N}_{eff}$ is small, perform resampling
  8. Increment t and return to step 3

Solving two different ways

- Solving for the filter distribution
- Solving for the posterior [degeneracy issue]

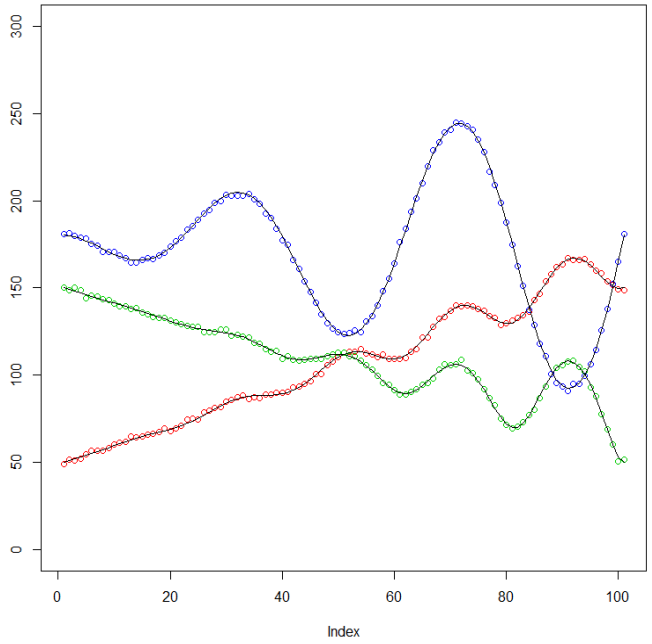
- Code GPS_EX.R

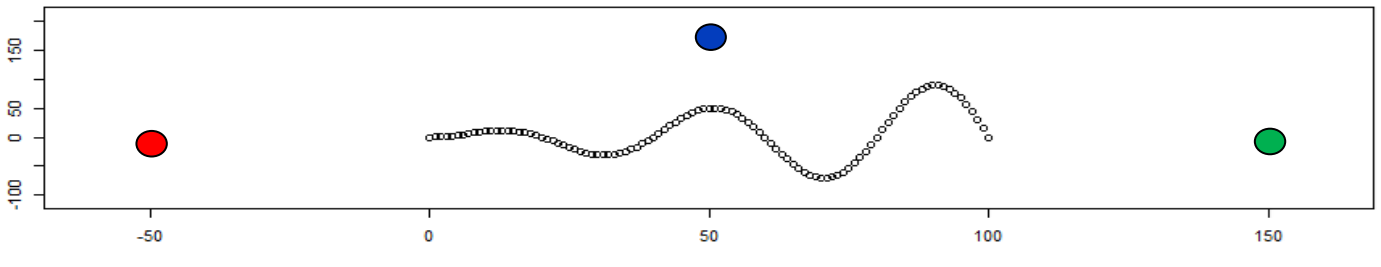


Data

Distance to blue
Distance to green

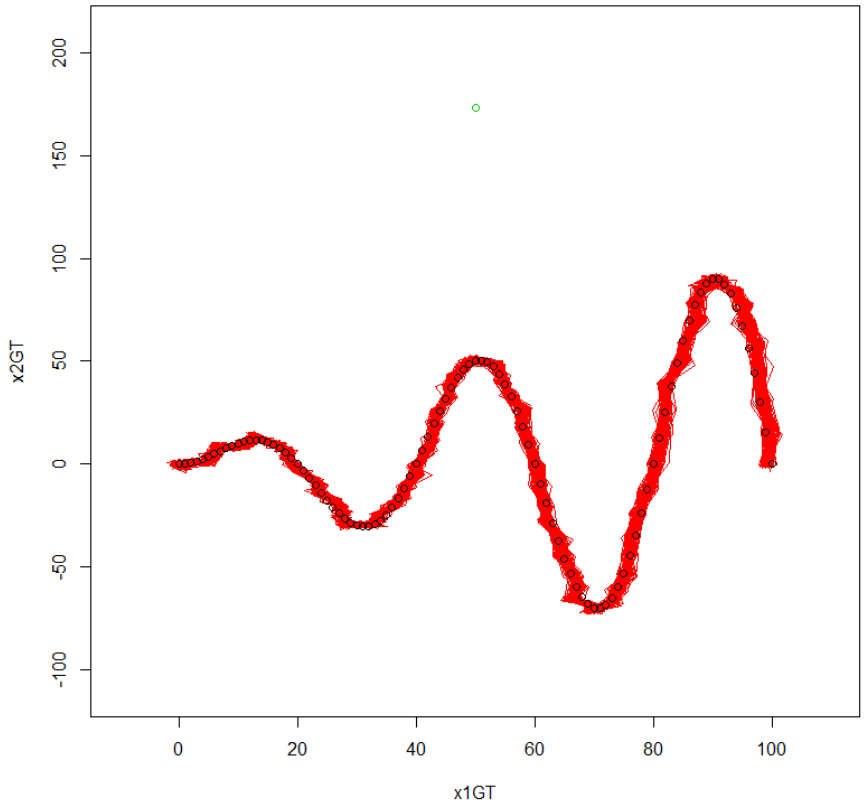
Distance to red

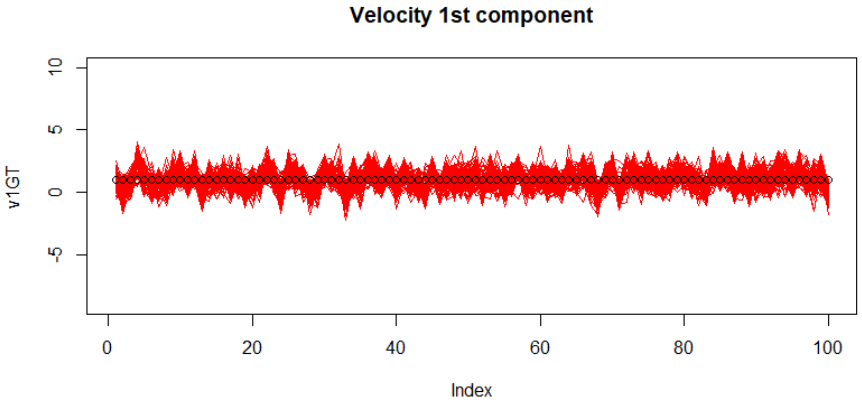


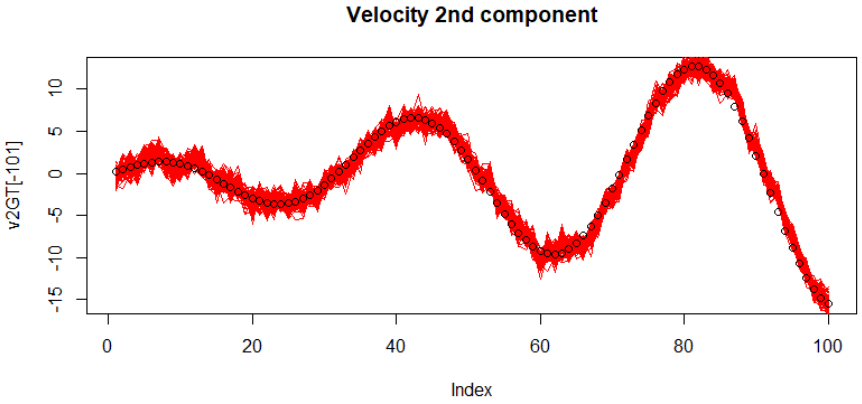


Filter solution

$$p(x_t, v_t | d_1, \ldots, d_t)$$



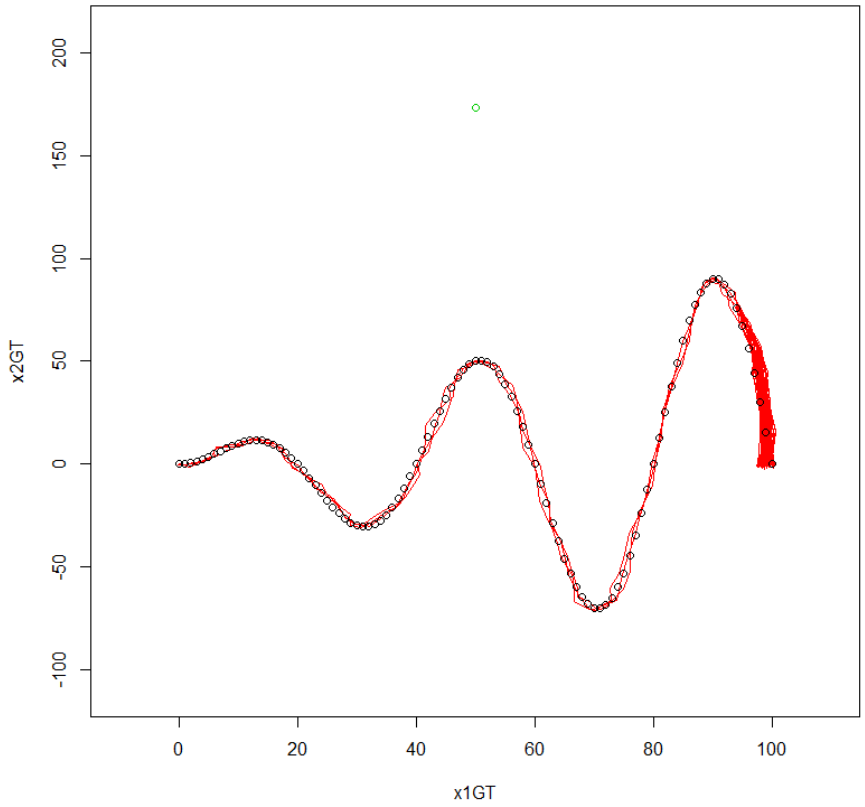


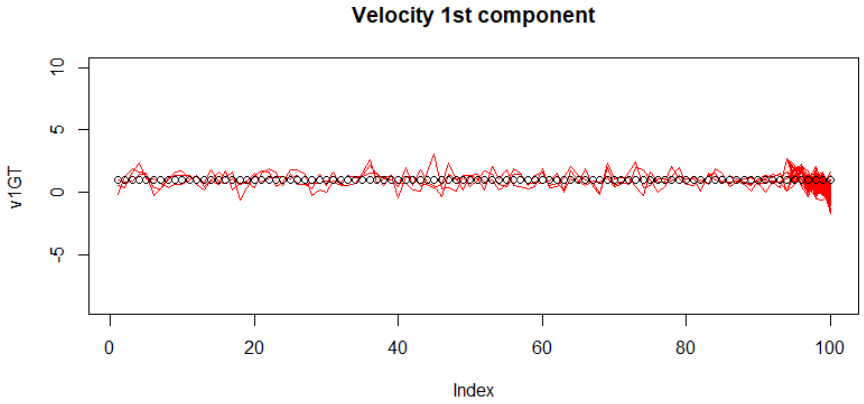


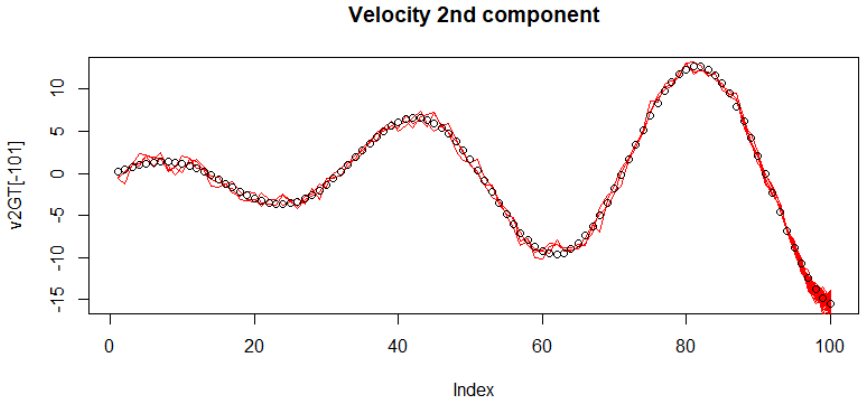
Smoothing «solution»

$$p(x_t, v_t | d_1, \ldots, d_t, \ldots, d_n)$$

Problem with Degeneracy !







Terrain navigation

- Assume movement model for airplane

Model

$$\mathbf{X}_t = \mathbf{x}_{t-1} + \mathbf{d}_t + \boldsymbol{\varepsilon}_t$$

$\mathbf{d}_t$ = Drift of plane measured by internal navigation system (assumed known)

$$\boldsymbol{\varepsilon}_t = \mathbf{R}_t^T \mathbf{Z}_t$$

$$\mathbf{R}_t = \frac{1}{\sqrt{x_{1,t-1}^2 + x_{2,t-1}^2}} \begin{pmatrix} -x_{1,t-1} & x_{2,t-1} \\ -x_{2,t-1} & -x_{1,t-1} \end{pmatrix}$$

$$\mathbf{Z}_t \sim N_2\left(\mathbf{0}, q^2 \begin{pmatrix} 1 & 0 \\ 0 & k^2 \end{pmatrix}\right)$$

Data

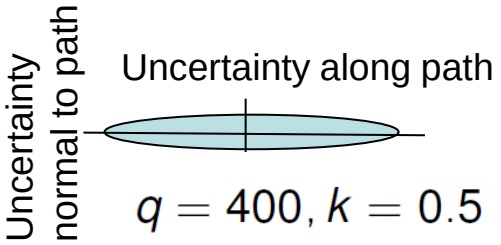
$$Y_t = m(\mathbf{x}_t) + \delta_t$$

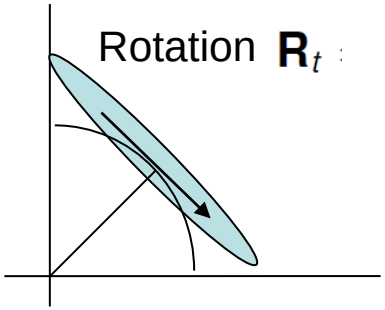
Map

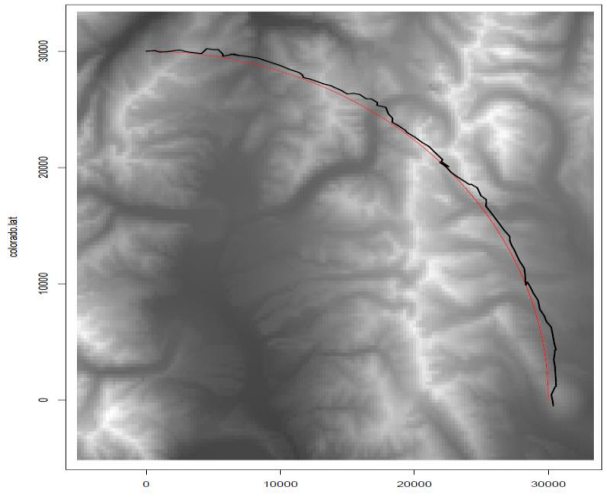
$m(\mathbf{x}_t)$ = Elevation at point $\mathbf{x}_t$

- `Example_6_7.R`

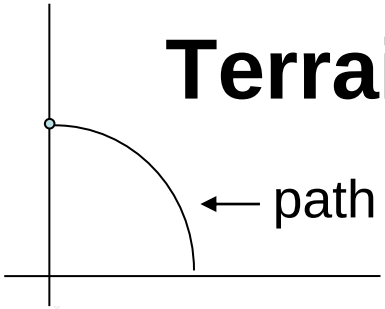
Uncertainty $\boldsymbol{\varepsilon}_t, \delta_t$







Terrain navigation




```
# n = number of sampled trajectories
# Yt = observed elevation data
# mxti = map elevations
# xt.x, xt.y = current position of point
# uti = weight adjustment factors
# wti = weights
# Neff = effective sample size
# alpha = rejuvenation trigger
# dsubt.x, dsubt.y = true drift
# epst.x, epst.y = location error
```

```
##INITIAL VALUES
n=100
sigma=75
q.value=400
k=.5
sdx0=50
sqrtP0=sdx0     #My change
wti=rep(1/n,n)
```

Initial ensemble members equal weight

```
##SET UP TRUE STARTING POINT, 100 INITIAL POINTS
##AND FIRST ELEVATION OBSERVATION
x0hat.x=0 ; x0hat.y=30000  #start at true X0 here

truex=x0hat.x
truey=x0hat.y
xthat.x=truex
xthat.y=truey
xt.x=rnorm(n,x0hat.x,sqrtP0)    #was x.xold
xt.y=rnorm(n,x0hat.y,sqrtP0)    #was y.yold
```

Initial ensemble (n = 100)

- Initialization
  - Path is constructed
  - Radius of curve 30 000

$\mathbf{d}_t$

```
##SET UP TRUE DRIFT
route.theta=seq(0,pi/2,length=101)
route.x=30000*cos(route.theta)
route.y=30000*sin(route.theta)
route.x=rev(route.x)
route.y=rev(route.y)
dsubt.x=diff(route.x)
dsubt.y=diff(route.y)
```

Just one way to get it from the synthetic

UiO : Matematisk institutt
Det matematisk-naturvitenskapelige fakultet

Terrain navigation

```
# n = number of sampled trajectories
# Yt = observed elevation data
# mxti = map elevations
# xt.x, xt.y = current position of point
# uti = weight adjustment factors
# wti = weights
# Neff = effective sample size
# alpha = rejuvenation trigger
# dsubt.x, dsubt.y = true drift
# epst.x, epst.y = location error
```

Mimic observation process

```
#update truth and observed elevation data
  truex=truex+dsubt.x[i]
  truey=truey+dsubt.y[i]
  Yt=interp(colo.lon.rep,colo.lat.rep,
       colo.elev.interp,xo=truex,yo=truey)$z+rnorm(1,0,sigma)
```

Update weights from data

```
#find m(x_t^i)
  xord=rank(xt.x)
  yord=rank(xt.y)
  mapt=interp(colo.lon.rep,colo.lat.rep,colo.elev.interp,
     xo=sort(xt.x),yo=sort(xt.y))
  mxti=mapt$z[cbind(xord,yord)]
  extrap=is.na(mxti)
#weight the points
  uti=ifelse(extrap,0,dnorm(Yt,c(mxti),rep(sigma,n)))
  wti=uti*wti
  wti=wti/sum(wti)
  Neff=1/sum(wti^2)
  neff.record[i]=Neff
```

Current estimate

```
#preliminary calcs for drawing the plot
  xthat.old=c(xthat.x,xthat.y)
  xthat.x=sum(wti*xt.x)
  xthat.y=sum(wti*xt.y)
```

$$\mathbf{X}_t = \mathbf{x}_{t-1} + \mathbf{d}_t + \boldsymbol{\varepsilon}_t$$

```
#update cloud
  tangent.slope=-truex/truey
  Zsigmamat=cbind(c(q.value^2,0),1*c(0,(k*q.value)^2))
  xtnext=rotnorm(n,tangent.slope,Zsigmamat)
  epst.x=xtnext[,1]
  epst.y=xtnext[,2]

rotnorm=function(N,slope,sigmamat) {
     v=rmvnorm(N,mean=c(0,0),sigma=sigmamat)
     xy=c(1,slope)
     Rot=cbind(c(-xy[1],-xy[2]),c(xy[2],-xy[1]))/sqrt(sum(xy^2))
     therot=t(Rot%*%t(v))
     therot }
```

Terrain navigation

```
# n = number of sampled trajectories
# Yt = observed elevation data
# mxti = map elevations
# xt.x, xt.y = current position of point
# uti = weight adjustment factors
# wti = weights
# Neff = effective sample size
# alpha = rejuvenation trigger
# dsubt.x, dsubt.y = true drift
# epst.x, epst.y = location error
```

Resample?

```
if (Neff<(alpha*n)) {
   idx=sample(1:n,n,replace=T,prob=wti)
   xtnew.x=xt.x[idx]
   xtnew.y=xt.y[idx]
   wti=rep(1/n,n)
   rejuvcount=rejuvcount+1
   reset=T }
```

Update according to dynamic model

```
if (!reset) {
   xtnext.x=xt.x+dsubt.x[i]+epst.x      #still using old points
   xtnext.y=xt.y+dsubt.y[i]+epst.y
} else {
   xtnext.x=xtnew.x+dsubt.x[i]+epst.x   #start with new points
   xtnext.y=xtnew.y+dsubt.y[i]+epst.y
   reset=F
}
xt.x=xtnext.x
xt.y=xtnext.y
```