Sampling Methods, Particle Filtering, and Markov-Chain Monte Carlo

CSE598C Vision-Based Tracking

Fall 2012, CSE Dept, Penn State Univ

References

A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking

M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp

INTRODUCTION TO MONTE CARLO METHODS

D.J.C. MACKAY

Department of Physics, Cambridge University.
Cavendish Laboratory, Madingley Road,
Cambridge, CB3 0HE. United Kingdom.

Markov Chain Monte Carlo for Computer Vision

A tutorial at the 10th Int'l Conf. on Computer Vision
October, 2005, Beijing

by

Song-Chun Zhu, UCLA
Frank Dellaert, Gatech
Zhuowen Tu, UCLA

Recall: Bayesian Filtering

Rigorous general framework for tracking. Estimates the values of a state vector based on a time series of uncertain observations.

Key idea: use a recursive estimator to construct the posterior density function (pdf) of the state vector at each time t based on all available data up to time t.

Bayesian hypothesis: All quantities of interest, such as MAP or marginal estimates, can be computed from the posterior pdf.

State Space Approach

Two vectors of interest:

1) State vector: vector of variables x_k representing what we want to know about the world

examples: [x,y]; [x,y,dx,dy]; [x,y,θ,scale]

2) Measurement vector: noisy observations z_k related to the state vector.

examples: image intensity/color; motion blobs

Robert Collins
CSE598C

Discrete Time Models

Discrete Time Formulation - measurements become available at discrete time steps 1,2,3,..,k,...
(very appropriate for video processing applications)

Need to specify two models:

1) System model - how current state is related to previous state (specifies evolution of state with time)

$x_k = f_k(x_{k-1}, v_{k-1})$ *v is process noise*

2) Measurement model - how noisy measurements are related to the current state

$z_k = h_k(x_k, n_k)$ *n is measurement noise*

Recursive Filter

We want to recursively estimate the current state at every time that a measurement is received.

<u>Two step approach:</u>

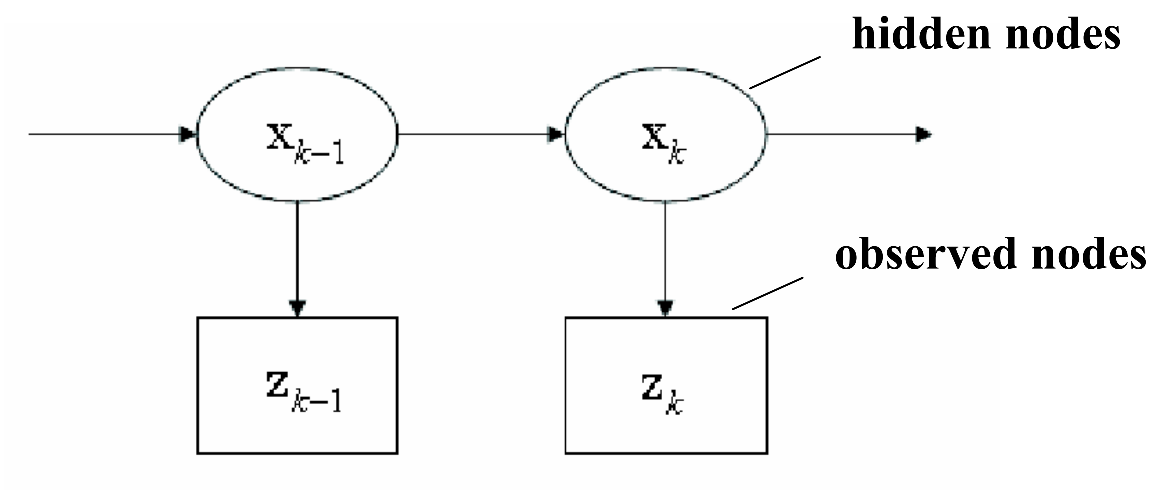
1) prediction: propagate state pdf forward in time, taking process noise into account (translate, deform, and spread the pdf)

2) update: use Bayes theorem to modify prediction pdf based on current measurement

Robert Collins
CSE598C

Tracking as a Graphical Model

Filtering as a hidden Markov model



Markov assumptions

$$p(\mathbf{x}_k|\mathbf{x}_0, \ldots, \mathbf{x}_{k-1}) = p(\mathbf{x}_k|\mathbf{x}_{k-1})$$

$$p(\mathbf{z}_k|\mathbf{x}_0, \ldots, \mathbf{x}_k) = p(\mathbf{z}_k|\mathbf{x}_k)$$

Factored joint probability distribution

$$p(\mathbf{x}_0, \ldots, \mathbf{x}_k, \mathbf{z}_1, \ldots, \mathbf{z}_k) = p(\mathbf{x}_0) \prod_{i=1}^{k} p(\mathbf{z}_i|\mathbf{x}_i) p(\mathbf{x}_i|\mathbf{x}_{i-1})$$

Recursive Bayes Filter

Motion Prediction Step:

$$\overbrace{p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}^{\text{predicted current state}} = \int \overbrace{p(\mathbf{x}_k|\mathbf{x}_{k-1})}^{\text{state transition}}\overbrace{p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})}^{\text{previous estimated state}}\, d\mathbf{x}_{k-1}.$$

Data Correction Step (Bayes rule):

$$\overbrace{p(\mathbf{x}_k|\mathbf{z}_{1:k})}^{\text{estimated current state}} = \frac{\overbrace{p(\mathbf{z}_k|\mathbf{x}_k)}^{\text{measurement}}\overbrace{p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}^{\text{predicted current state}}}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})}$$

normalization term

$$p(\mathbf{z}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})\, d\mathbf{x}_k$$

Problem

Except in special cases, these integrals are intractible.

Motion Prediction Step:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \boxed{\int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})\, d\mathbf{x}_{k-1}}.$$

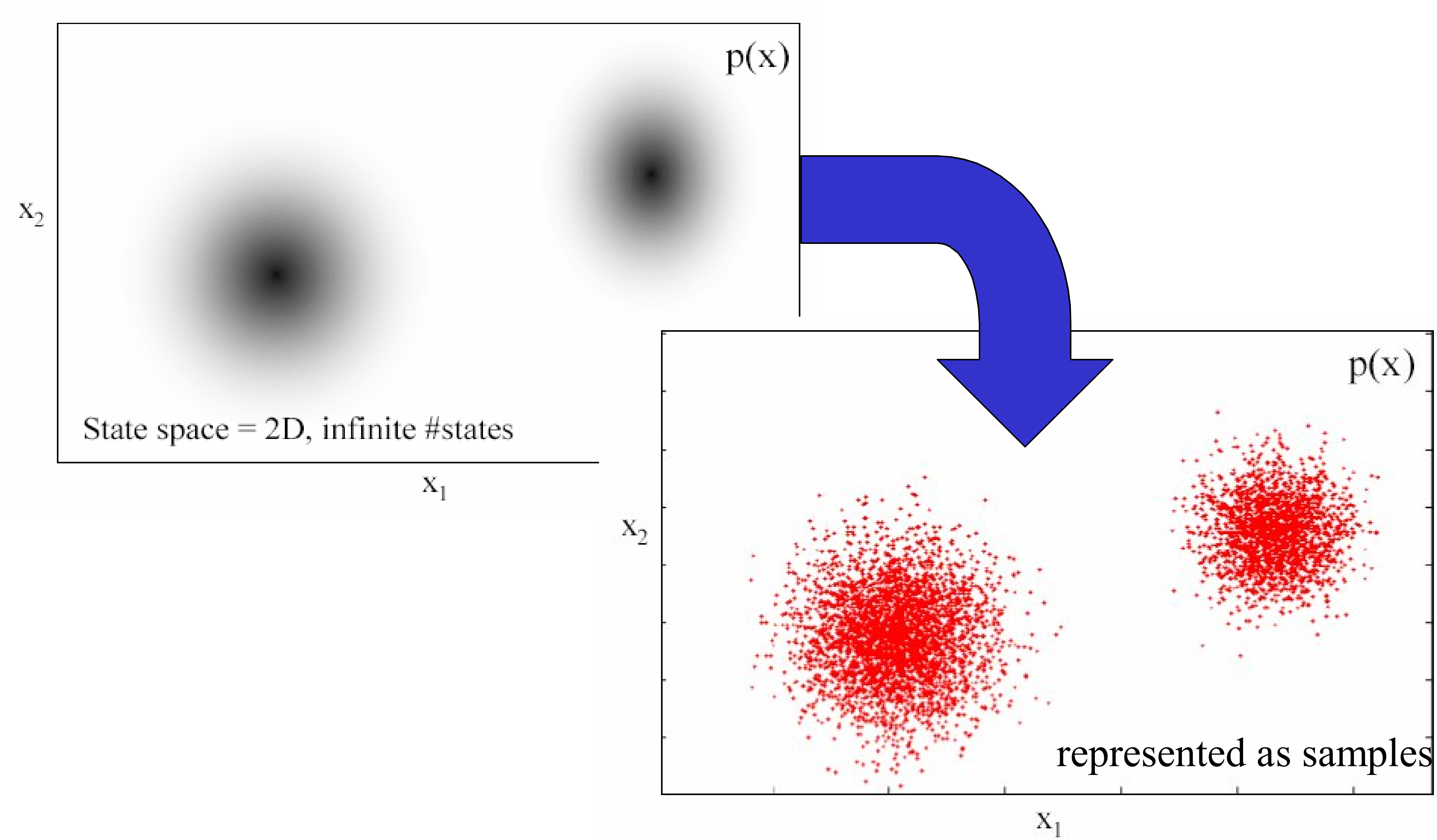
Data Correction Step (Bayes rule):

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})}$$

$$p(\mathbf{z}_k|\mathbf{z}_{1:k-1}) = \boxed{\int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})\, d\mathbf{x}_k}$$

(examples of special cases: linear+Gaussian (Kalman); discrete state spaces)

Idea: Represent PDFs by Samples



related idea: Parzen Estimation

Why Does This Help?

If we can generate random samples $\mathbf{x}_i$ from a given distribution P(x), then we can estimate expected values of functions under this distribution by summation, rather than integration.

That is, we can approximate:

$$E(f(x)) = \int f(x)P(x)dx$$

by first generating N i.i.d. samples from P(x) and then forming the empirical estimate:

$$\hat{E}(f(x)) = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

related idea: Monte Carlo Integration

A Brief Overview of Sampling

Inverse Transform Sampling (CDF)

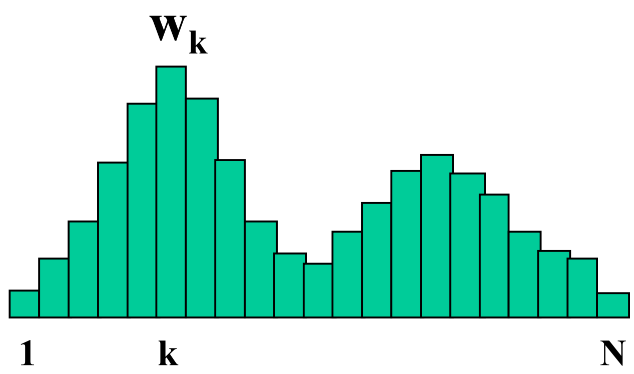
Rejection Sampling

Importance Sampling

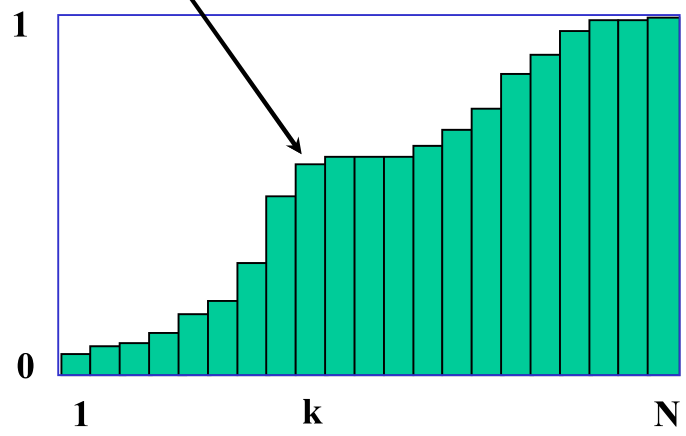
Robert Collins
CSE598C

Inverse Transform Sampling

It is easy to sample from a discrete 1D distribution, using the cumulative distribution function.



$$c(k) = \sum_{i=1}^{k} w_i / \sum_{i=1}^{N} w_i$$



cumulative distribution function

$$F(x) = P(X \leq x)$$

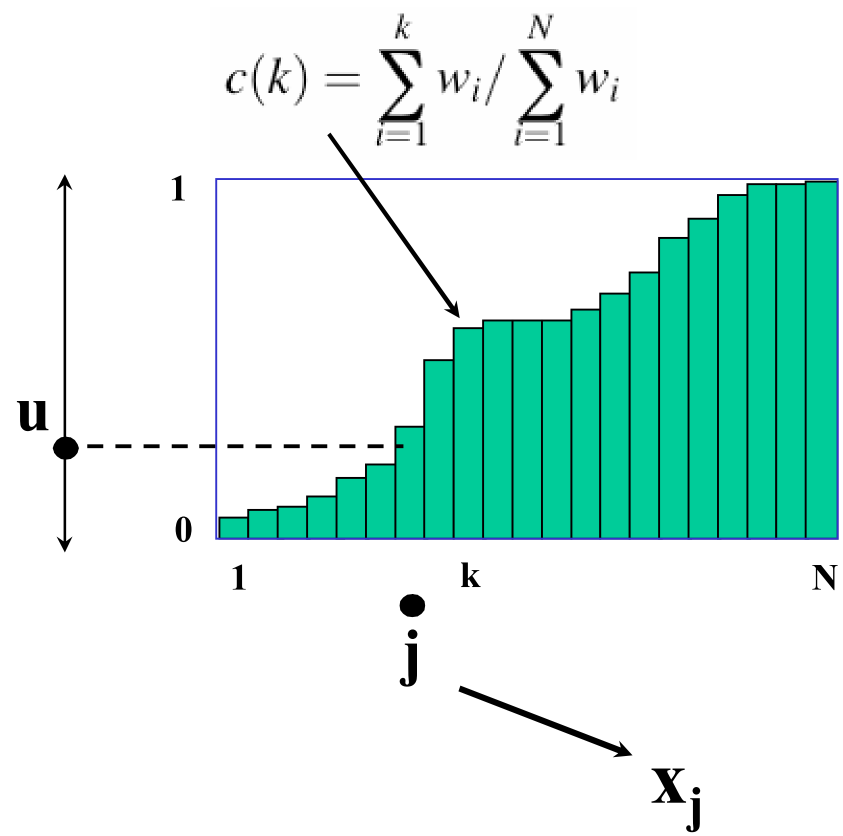
Inverse Transform Sampling

It is easy to sample from a discrete 1D distribution, using the cumulative distribution function.

1) **Generate uniform u in the range [0,1]**

2) **Visualize a horizontal line intersecting bars**

3) **If index of intersected bar is j, output new sample x_j**



Inverse Transform Sampling

Why it works:

cumulative distribution function

$$F(x) = P(X \le x)$$

inverse cumulative distribution function

$$F^{-1}(t) = \min\{x : F(x) = t, 0 < t < 1\}$$

Claim: if U is a uniform random variable on $(0,1)$ then X=$F^{-1}(U)$ has distribution function F.
Proof:

$$\begin{aligned} & P(F^{-1}(U) \le x) \\ = \;& P(\min\{x : F(x) = U\} \le x) && \text{(def of } F^{-1}) \\ = \;& P(U \le F(x)) && \text{(applied } F \text{ to both sides)} \\ = \;& F(x) && \text{(def of distribution function of } U) \end{aligned}$$

Efficient Generating Many Samples

from Arulampalam paper

Algorithm 2: Resampling Algorithm

$[\{\mathbf{x}_k^{j*}, w_k^j, i^j\}_{j=1}^{N_s}] = \text{RESAMPLE}\ [\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}]$

- Initialize the CDF: $c_1 = 0$
- FOR $i = 2: N_s$
  - Construct CDF: $c_i = c_{i-1} + w_k^i$
- END FOR
- Start at the bottom of the CDF: $i = 1$
- Draw a starting point: $u_1 \sim \mathbb{U}[0, N_s^{-1}]$
- FOR $j = 1: N_s$
  - Move along the CDF: $u_j = u_1 + N_s^{-1}(j-1)$
  - WHILE $u_j > c_i$
    - * $i = i + 1$
  - END WHILE
  - Assign sample: $\mathbf{x}_k^{j*} = \mathbf{x}_k^i$
  - Assign weight: $w_k^j = N_s^{-1}$
  - Assign parent: $i^j = i$
- END FOR

Basic idea: choose one initial small random number; deterministically sample the rest by "crawling" up the cdf function. This is O(N).

odd property: you generate the "random" numbers in sorted order...

A Brief Overview of Sampling

Inverse Transform Sampling (CDF)

Rejection Sampling

Importance Sampling

For these two, we can sample from continuous distributions, and they do not even need to be normalized.

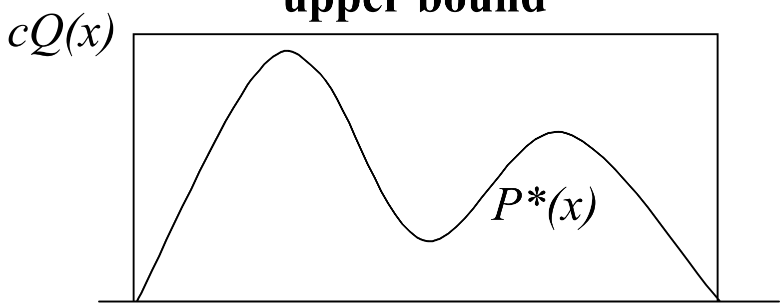
That is, to sample from distribution P, we only need to know a function P*, where P = P* / c , for some normalization constant c.

Rejection Sampling

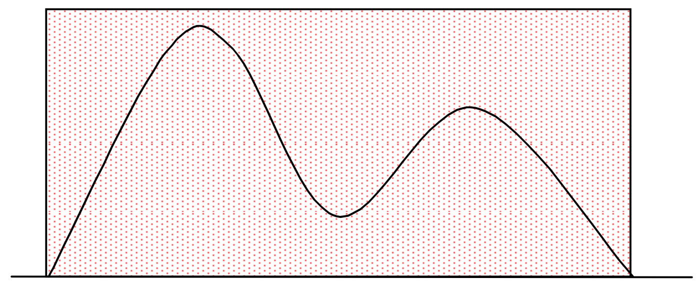
Need a proposal density Q(x) [e.g. uniform or Gaussian], and a constant c such that c(Qx) is an <u>upper bound</u> for P*(x)

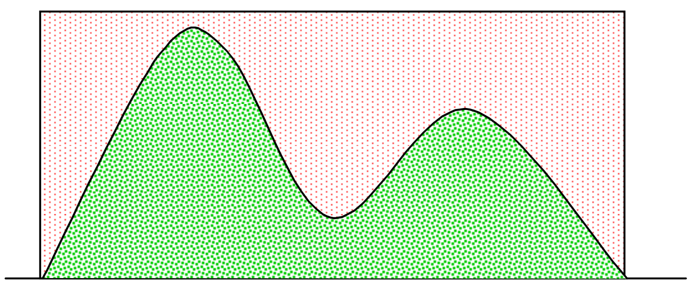
Example with Q(x) uniform

upper bound



generate uniform random samples in upper bound volume





accept samples that fall below the P*(x) curve

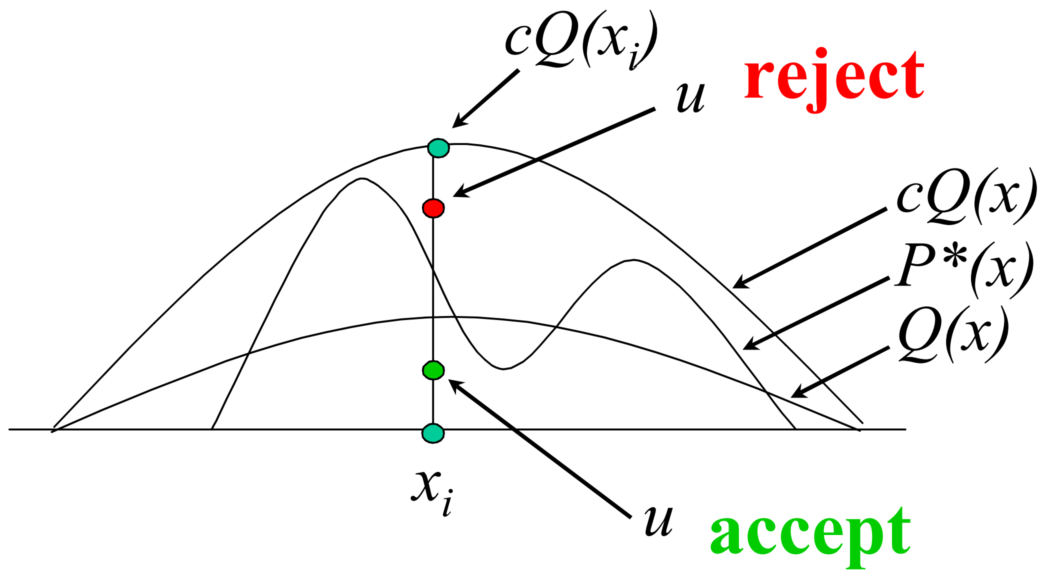
the marginal density of the x coordinates of the points is then proportional to P*(x)

Note: this very related to Monte Carlo integration.

Rejection Sampling

More generally:

1) generate sample x_i from a proposal density Q(x)
2) generate sample u from uniform [0,cQ(x_i)]
3) if u <= P*(x_i) accept x_i; else reject



Importance "Sampling"

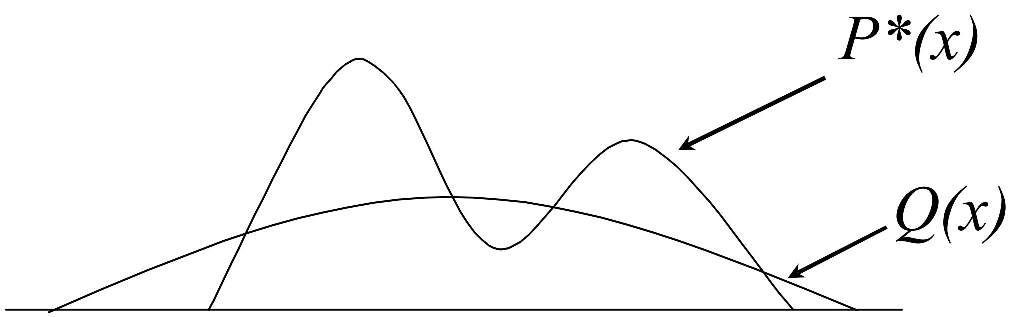
Not a method for generating samples. It is a method for estimating expected value of functions $f(x_i)$

1) Generate x_i from $Q(x)$
2) an empirical estimate of $E_Q(f(x))$, the expected value of $f(x)$ under distribution $Q(x)$, is then

$$\hat{E}_Q(f(x)) = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

3) However, we want $E_P(f(x))$, which is the expected value of $f(x)$ under distribution $P(x) = P^*(x)/Z$

Importance Sampling



When we generate from Q(x), values of x where Q(x) is greater than P*(x) are overrepresented, and values where Q(x) is less than P*(x) are underrepresented.

To mitigate this effect, introduce a weighting term

$$w_i = \frac{P^*(x_i)}{Q(x_i)}$$

Importance Sampling

New procedure to estimate $E_P(f(x))$:

1) Generate N samples x_i from Q(x)

2) form importance weights

$$w_i = \frac{P^*(x_i)}{Q(x_i)}$$

3) compute empirical estimate of $E_P(f(x))$, the expected value of f(x) under distribution P(x), as

$$\hat{E}_P(f(x)) = \frac{\sum w_i f(x_i)}{\sum w_i}$$

Resampling

Note: We thus have a set of weighted samples (xi, wi | i=1,…,N)

If we really need random samples from P, we can generate them by resampling such that the likelihood of choosing value x_i is proportional to its weight w_i

This would now involve now sampling from a discrete distribution of N possible values (the N values of x_i)

Therefore, regardless of the dimensionality of vector x, we are resampling from a 1D distribution (we are essentially sampling from the indices 1...N, in proportion to the importance weights w_i). So we can using the inverse transform sampling method we discussed earlier.

Note on Proposal Functions

Computational efficiency is best if the proposal distribution looks a lot like the desired distribution (area between curves is small).

These methods can fail badly when the proposal distribution has 0 density in a region where the desired distribution has non-negligeable density.

For this last reason, it is said that the proposal distribution should have heavy tails.

Sequential Monte Carlo Methods

Sequential Importance Sampling (SIS) and the closely related algorithm Sampling Importance Sampling (SIR) are known by various names in the literature:

- bootstrap filtering
- particle filtering
- Condensation algorithm
- survival of the fittest

General idea: Importance sampling on time series data, with samples and weights updated as each new data term is observed. Well-suited for simulating recursive Bayes filtering!

Sequential Monte Carlo Methods

Intuition: each particle is a “guess” at the true state. For each one, simulate it’s motion update and add noise to get a motion prediction. Measure the likelihood of this prediction, and weight the resulting particles proportional to their likelihoods.

Back to Bayes Filtering

remember our intractible integrals:

Motion Prediction Step:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \boxed{\int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})\, d\mathbf{x}_{k-1}.}$$

Data Correction Step (Bayes rule):

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})}$$

$$p(\mathbf{z}_k|\mathbf{z}_{1:k-1}) = \boxed{\int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})\, d\mathbf{x}_k}$$

Back to Bayes Filtering

This integral in the denominator of Bayes rule goes away for free, as a consequence of representing distributions by a weighted set of samples. Since we have only a finite number of samples, we can easily compute the normalization constant by summing the weights!

Data Correction Step (Bayes rule):

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})}$$

$$p(\mathbf{z}_k|\mathbf{z}_{1:k-1}) = \boxed{\int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})\, d\mathbf{x}_k}$$

Back to Bayes Filtering

Now let's write the Bayes filter by combining motion prediction and data correction steps into one equation.

$$\underbrace{P(X_t|Z^t)}_{\text{new posterior}} = c\,\underbrace{P(Z_t|X_t)}_{\text{data term}} \int \underbrace{P(X_t|X_{t-1})}_{\text{motion term}}\,\underbrace{P(X_{t-1}|Z^{t-1})}_{\text{old posterior}}\,dX_{t-1}$$

Monte Carlo Bayes Filtering

Assume the posterior at time t-1 has been approximated as a set of N weighted particles:

$$P(X_{t-1}|Z^{t-1}) \approx \{X_{t-1}^{(r)}, \pi_{t-1}^{(r)}\}_{r=1}^{N},$$

The integral in Bayes filter

$$P(X_t|Z^t) = cP(Z_t|X_t) \int P(X_t|X_{t-1}) P(X_{t-1}|Z^{t-1}) dX_{t-1}$$

can then be computed by the Monte Carlo Approximation

$$P(X_t|Z^t) \approx cP(Z_t|X_t) \sum_r \pi_{t-1}^{(r)} P(X_t|X_{t-1}^{(r)})$$

Robert Collins
CSE598C

Representing the New Posterior

Now let's say we want to draw samples from this new posterior

$$P(X_t|Z^t) \approx cP(Z_t|X_t) \sum_r \pi_{t-1}^{(r)} P(X_t|X_{t-1}^{(r)})$$

We can do this by importance sampling, where we draw samples from a proposal distribution q(Xt) that is a mixture density

$$X_t^{(s)} \sim q(X_t) \triangleq \sum_r \pi_{t-1}^{(r)} P(X_t|X_{t-1}^{(r)})$$

The importance weights are computed as

$$\pi_t^{(s)} = \frac{P*}{Q} = \frac{P(Z_t|X_t) \sum_r \pi_{t-1}^{(r)} P(X_t|X_{t-1}^{(r)})}{\sum_r \pi_{t-1}^{(r)} P(X_t|X_{t-1}^{(r)})} = P(Z_t|X_t^{(s)})$$

To get a new weighted set of samples approximating the posterior at time t $P(X_t|Z^t) \sim \{X_t^{(s)}, \pi_t^{(s)}\}_{s=1}^N$

Note

- the previous interpretation based on importance sampling from a mixture model is nonstandard, and due to Frank Dellaert.

- A more traditional interpretation can be found in the Arulampalam tutorial paper.

SIS Algorithm

Algorithm 1: SIS Particle Filter

$[\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}] = \text{SIS}[\{\mathbf{x}_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, \mathbf{z}_k]$

- FOR $i = 1: N_s$
  - Draw $\mathbf{x}_k^i \sim q(\mathbf{x}_k|\mathbf{x}_{k-1}^i, \mathbf{z}_k)$
  - Assign the particle a weight, w_k^i, according to

$$w_k^i \propto w_{k-1}^i \frac{p(\mathbf{z}_k|\mathbf{x}_k^i)p(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i)}{q(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i, \mathbf{z}_k)}$$

- END FOR

SIR is then derived as a special case of SIS where
1) importance density is the prior density
2) resampling is done at every time step so that all the sample weights are 1/N.

SIR Algorithm

Algorithm 4: SIR Particle Filter

$[\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}] = \text{SIR}[\{\mathbf{x}_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, \mathbf{z}_k]$

- FOR $i = 1 : N_s$
  - Draw $\mathbf{x}_k^i \sim p(\mathbf{x}_k | \mathbf{x}_{k-1}^i)$
  - Calculate $w_k^i = p(\mathbf{z}_k | \mathbf{x}_k^i)$
- END FOR
- Calculate total weight: $t = \text{SUM}[\{w_k^i\}_{i=1}^{N_s}]$
- FOR $i = 1 : N_s$
  - Normalize: $w_k^i = t^{-1} w_k^i$
- END FOR
- Resample using algorithm 2:
  - $[\{\mathbf{x}_k^i, w_k^i, -\}_{i=1}^{N_s}] = \text{RESAMPLE}[\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}]$

Drawing from the Prior Density

note, when we use the prior as the importance density, we only need to sample from the process noise distribution (typically uniform or Gaussian).

Why? Recall:

$$x_k = f_k\,(x_{k-1}, v_{k-1}) \qquad v \text{ is process noise}$$

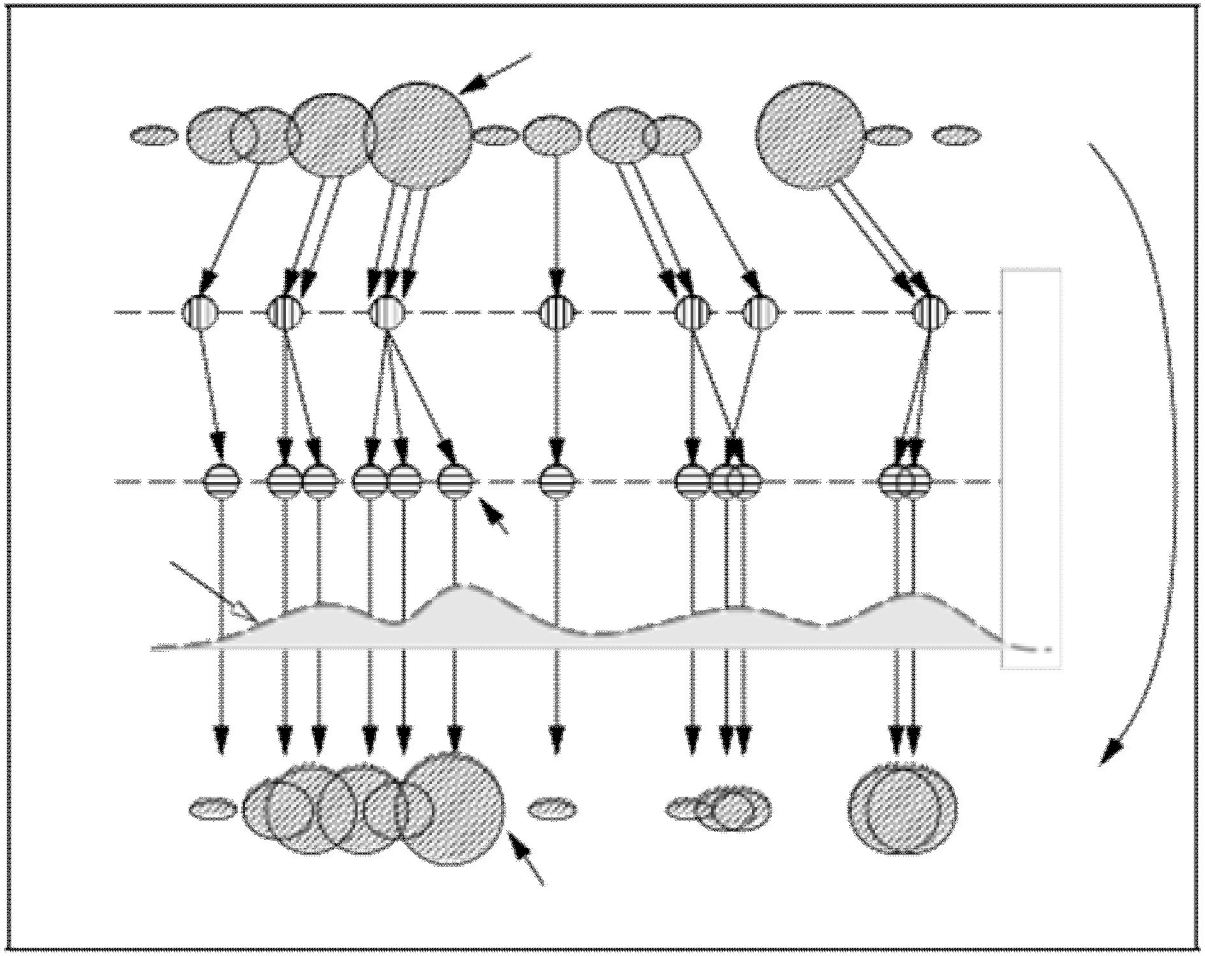
Thus we can sample from the prior $P(x_k \mid x_{k-1})$ by generating a sample x^i_{k-1}, generating a noise vector v^i_{k-1} from the noise process, and forming the noisy sample

$$x^i_k = f_k\,(x^i_{k-1}, v^i_{k-1})$$

If the noise is additive, this leads to a very simple interpretation: move each particle using motion prediction, then add noise.

Condensation (Isard&Blake)

time t-1



time t

normalized set of weighted samples

draw samples and apply motion predict

add noise (diffusion)

weight each sample by the likelihood

renormalize to get new set of samples

Problems with SIS/SIR

Degeneracy: in SIS, after several iterations all samples except one tend to have negligible weight. Thus a lot of computational effort is spent on particles that make no contribution. Resampling is supposed to fix this, but also causes a problem...

Sample Impoverishment: in SIR, after several iterations all samples tend to collapse into a single state. The ability to representation multimodal distributions is thus short-lived.

Next Time

We will look at the SIR sample impoverishment problem in detail, by viewing the algorithm's behavior as a Markov Chain.

This is fruitful for two reasons.

1) understanding the limitations of SIR/Condensation

2) introduce ideas that form the basis of Markov Chain Monte Carlo (MCMC), which solves these problems.