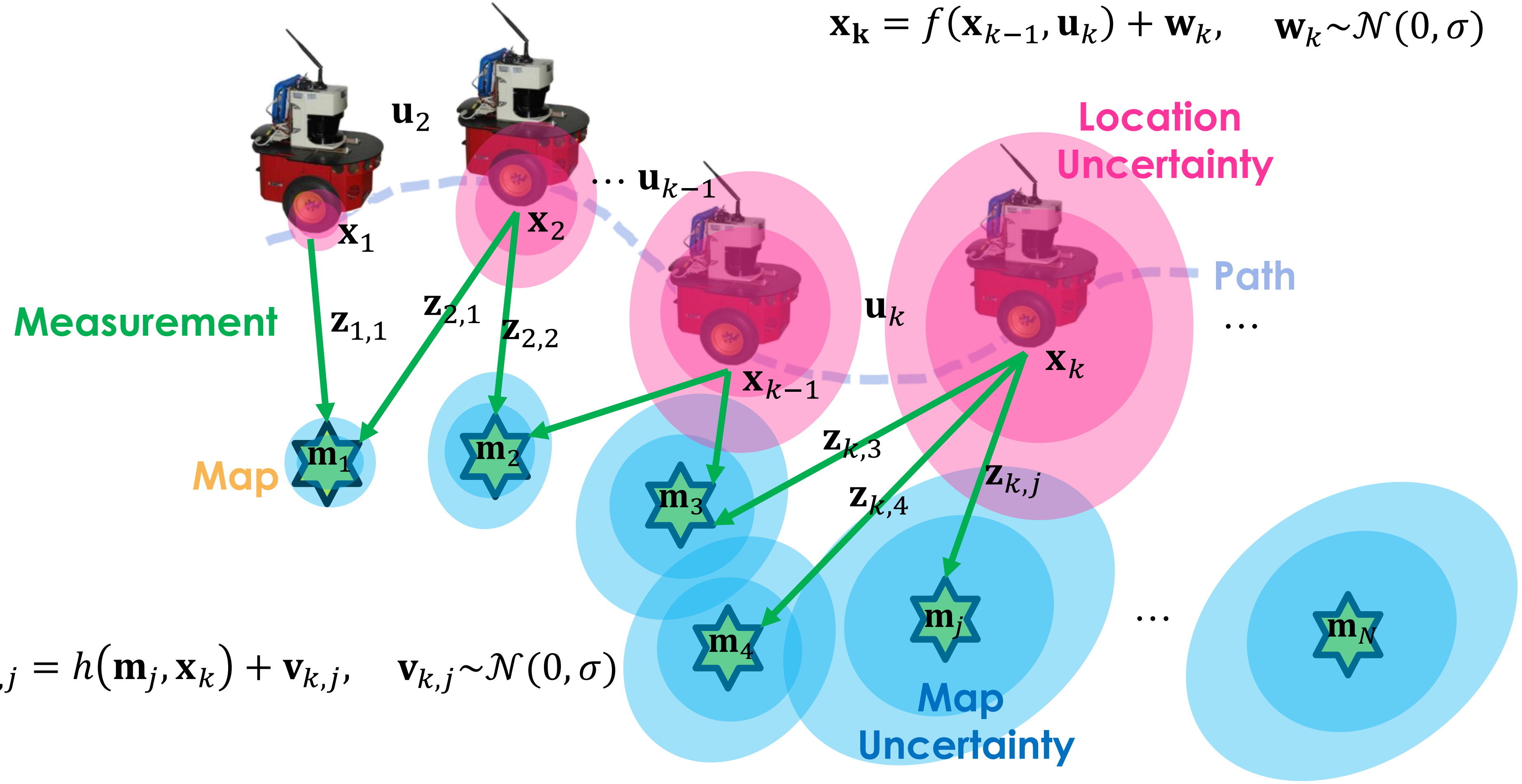
Robotic Navigation and Exploration

Week 5: SLAM Back-end (II)

Min-Chun Hu anitahu@cs.nthu.edu.tw
CS, NTHU

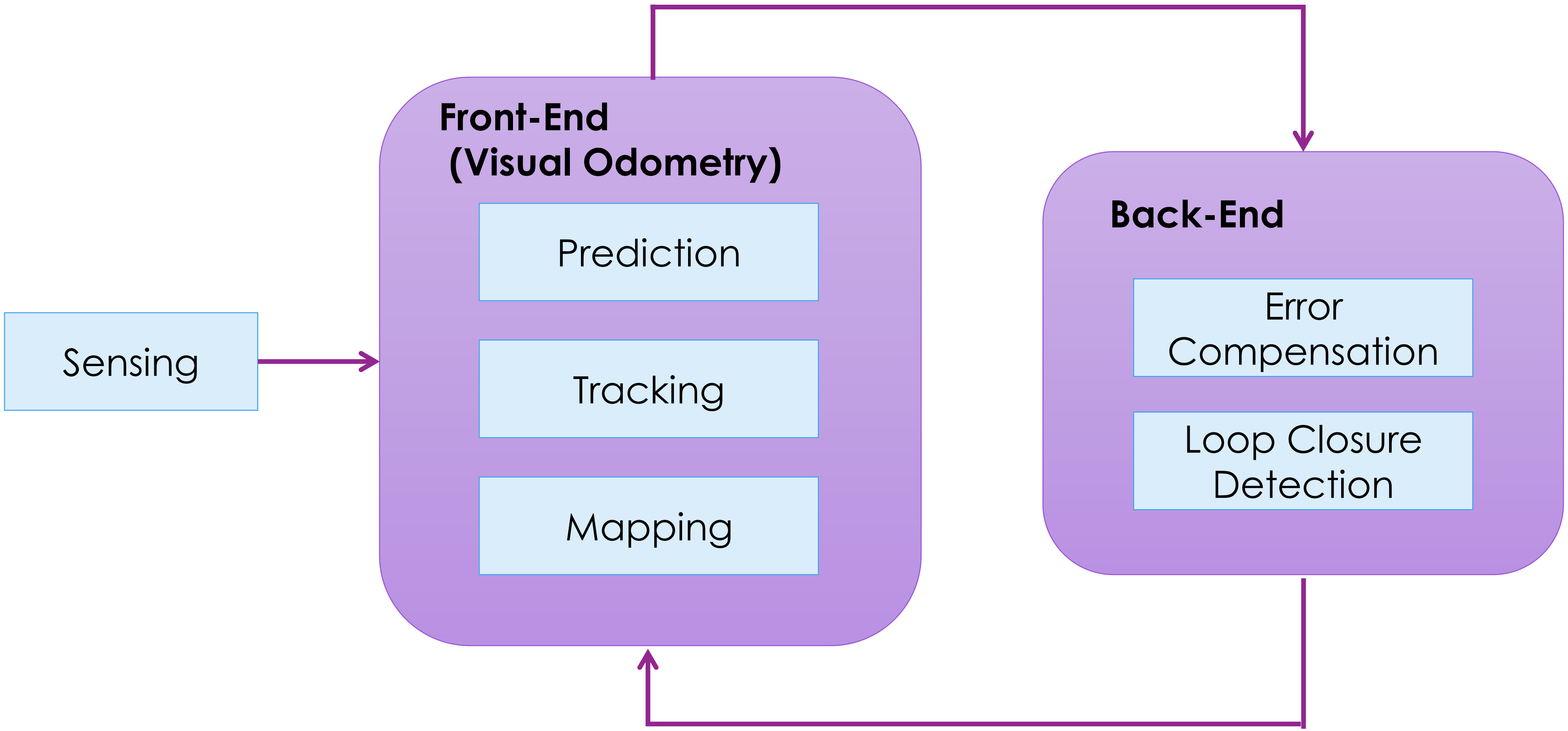
Recall the SLAM Problem

$$\mathbf{x_k} = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k, \quad \mathbf{w}_k \sim \mathcal{N}(0, \sigma)$$



$$\mathbf{z}_{k,j} = h(\mathbf{m}_j, \mathbf{x}_k) + \mathbf{v}_{k,j}, \quad \mathbf{v}_{k,j} \sim \mathcal{N}(0, \sigma)$$

SLAM Architecture



Error Compensation Methods

- Filter-based
  - Small Computation
  - On-line Optimization

- Graph-based
  - Large computation
  - High Accuracy
  - Off-line Optimization

Outline

- State Estimation and SLAM Problem
- SLAM Back-end (Error Compensation)
  - Filter-based Methods
    - Probability Theory and Bayes Filter
    - Kalman Filter (KF) / Extended Kalman Filter (EKF)
      - EKF-SLAM
    - Particle Filter
      - Fast-SLAM
  - Graph-based Methods
    - Pose Graph and Least-square Optimization
    - Gauss-Newton and Levenberg-Marquardt Algorithm
    - Sparse Matrix for Optimization

Introduction to Particle Filter

- EKF-SLAM assumes the probability distribution of robot pose and landmarks to be Gaussian, which leads to the following drawbacks:
  - Gaussian distribution can not express the robot pose properly.
  - The time complexity of estimating the covariance matrix for pose and landmarks is ($O(K^2)$), which is time-consuming even when only observing few landmarks.

    (K: number of landmarks)
- Particle filter utilizes **importance sampling** to approximate arbitrary distribution, which can express the robot pose more precisely.
- Furthermore, the time complexity of posterior estimation can be decreased to **O(MlogK)** by disentangling the estimation process of pose and map.

Sampling Process

- In statistical modeling and inference, there are many complex problems that the closed-form descriptions of $\boldsymbol{P}(\boldsymbol{X})$ can not be obtained.
- One can define a function $f(X)$ that computes $P(X)$ up to a normalizing constant:

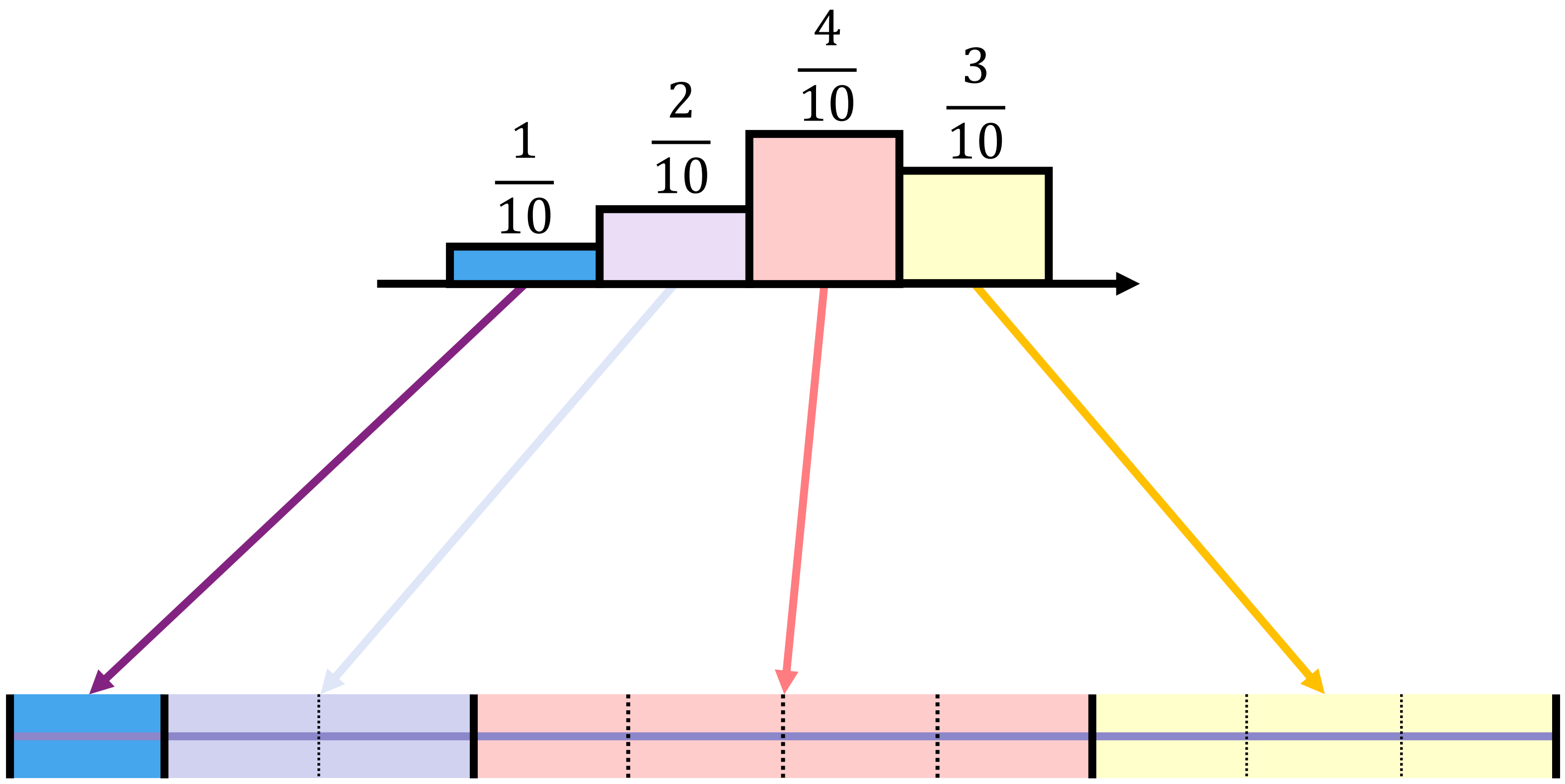
$$p(X) = \frac{f(X)}{Z}$$

where $z = \int_{x \in s} f(x)dx$ can not be computed because $f(X)$ is too complex, or because the state space $\boldsymbol{S}$ is too large to compute the integral.

- Statistical sampling and simulation techniques are used for getting fair samples from target probability distributions.

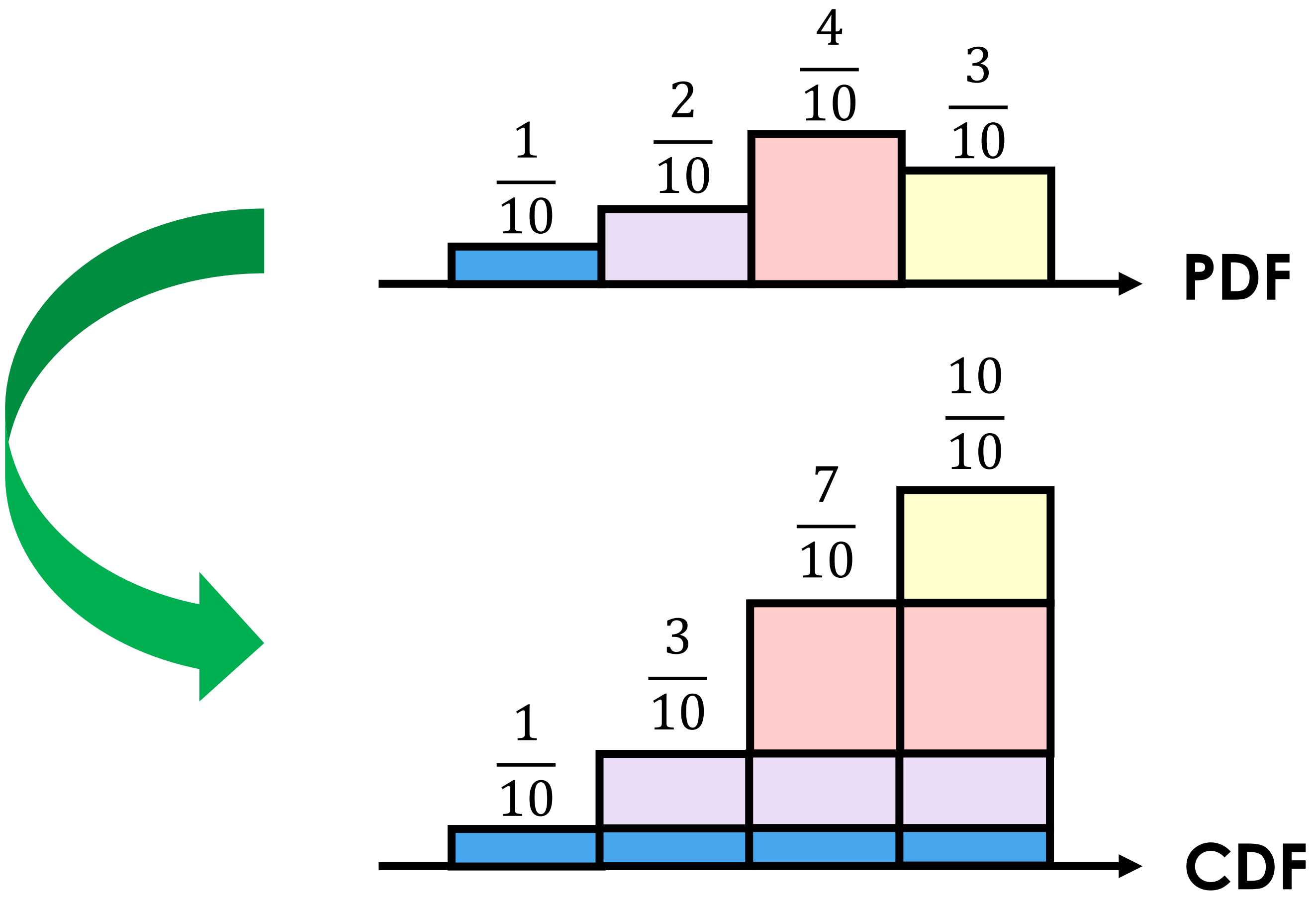
Basic Sampling

- Sampling from Probability Distribution Figure (PDF)



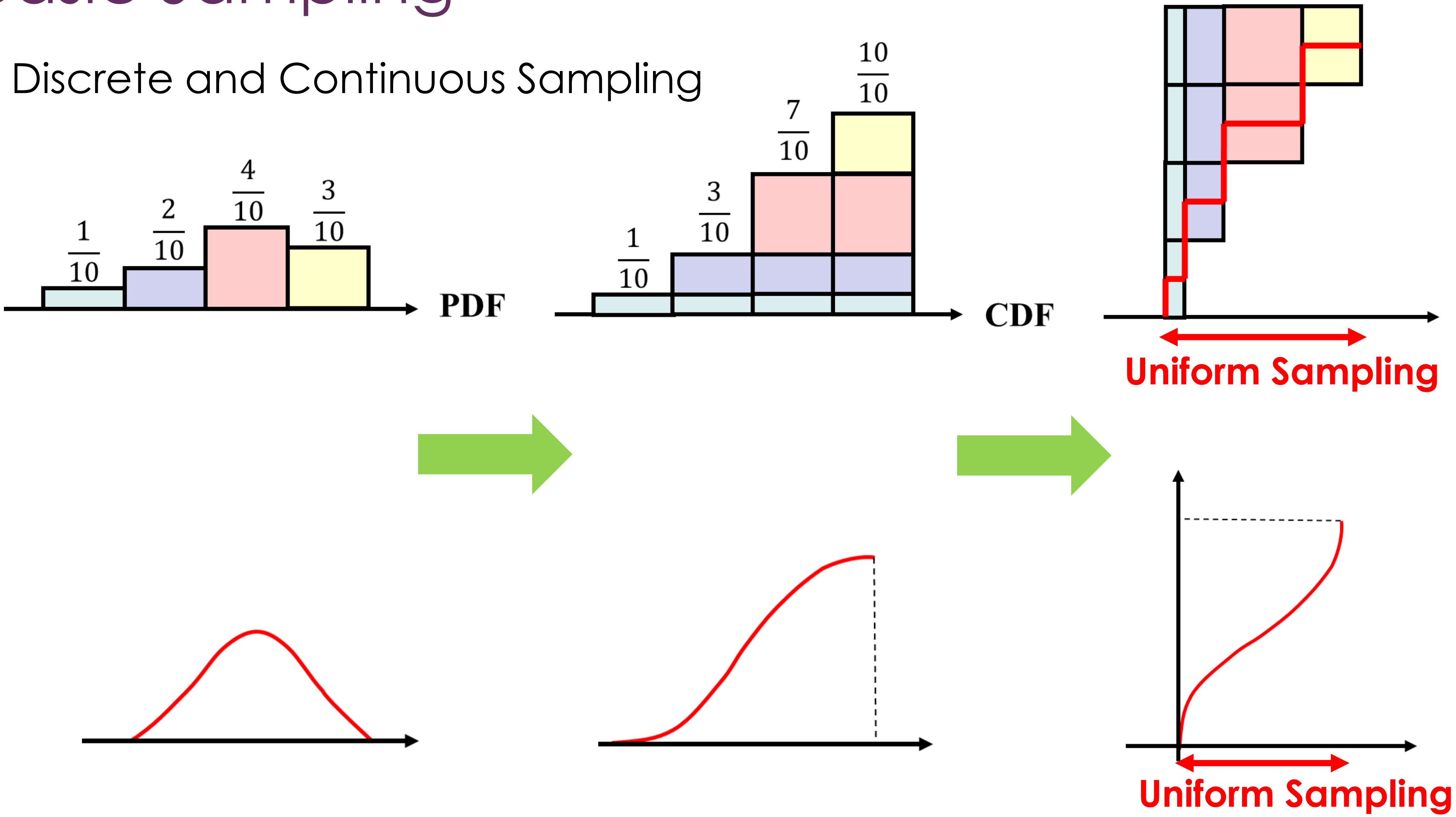
Basic Sampling

- From Probability Distribution Figure (PDF) to Cumulated Distribution Figure (CDF)



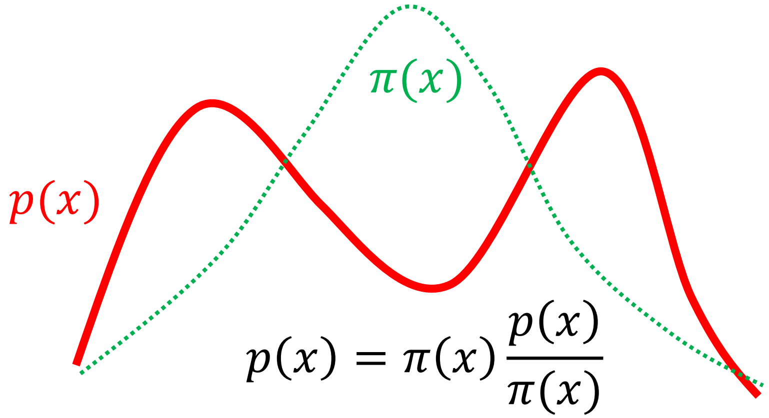
Basic Sampling

- Discrete and Continuous Sampling



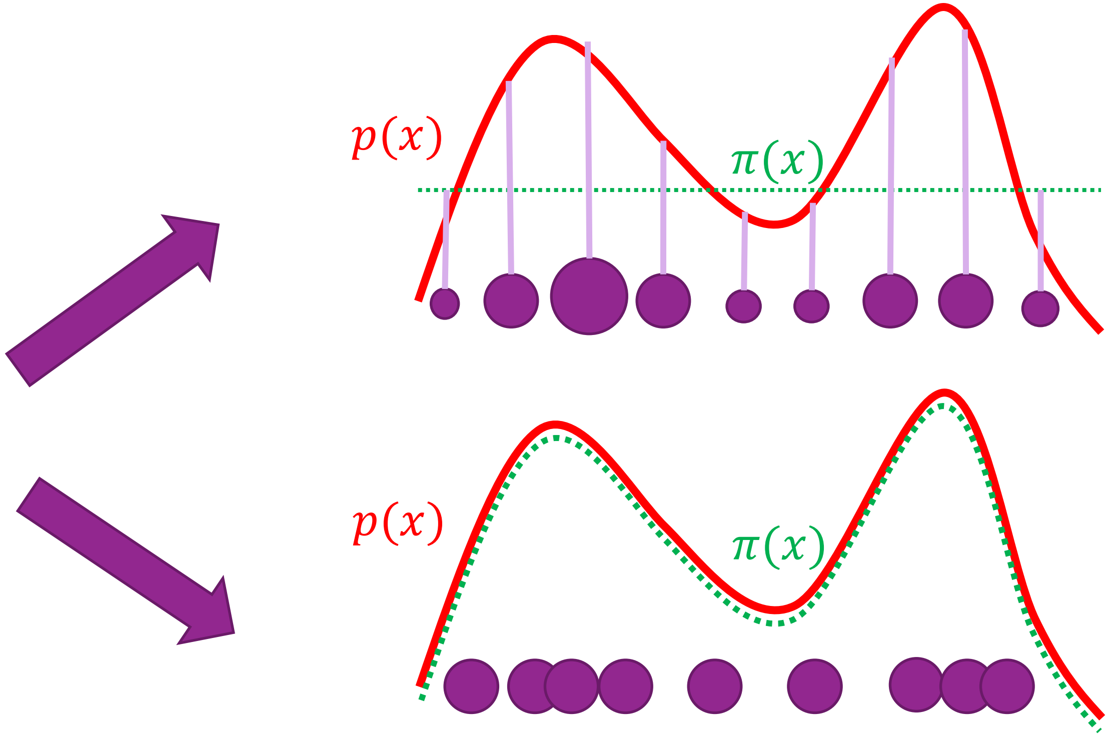
Importance Sampling

- Important sampling adopts discrete multinomial to approximate arbitrary distribution. More sampling particles will have more accurate approximation.



$$p(x) = \pi(x)\frac{p(x)}{\pi(x)}$$

1. $Sampling\ x_i\ from\ \pi(x)$
2. $Calculate\ w_i = \frac{p(x_i)}{\pi(x_i)}$
3. $Sampling\ x\ from\ mul(x_i, w_i)$

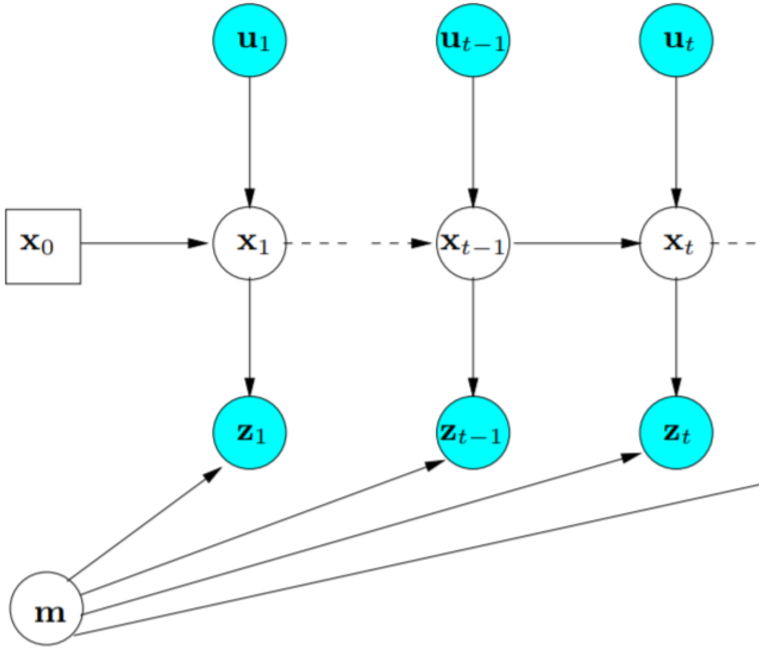


Sequential Importance Sampling (SIS)

- Consider the localization problem, we utilize several particles to represent the approximation of pose distribution.
- In importance sampling, each particle have its own pose and weighting. The weighting is the division of source distribution and target distribution:

$$w_t^{(i)} = \frac{p(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})}$$

- According to the graphical model, we have



$$w_t^{(i)} = \frac{p\left(x_t^{(i)}\middle|x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t-1}\right)}{\pi(x_t^{(i)}|x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t-1})} \cdot \frac{p\left(x_{1:t-1}^{(i)}\middle|z_{1:t-1}, u_{1:t-2}\right)}{\pi\left(x_{1:t-1}^{(i)}\middle|z_{1:t-1}, u_{1:t-2}\right)}$$

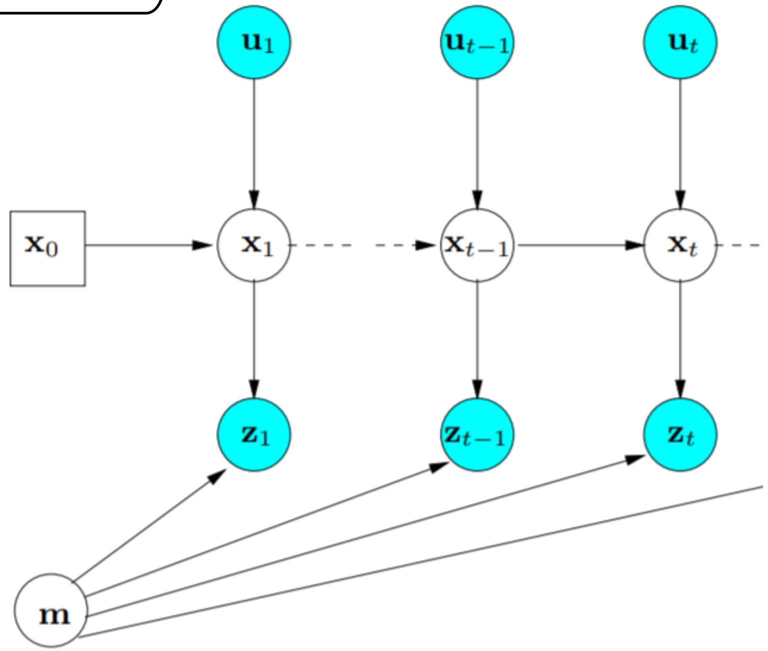
Sequential Importance Sampling (SIS)

- Apply the Bayes theorem, we can get

$$w_t^{(i)} = \frac{\eta p\left(z_t \middle| x_{1:t}^{(i)}, u_{1:t-1}\right) p\left(x_t^{(i)} \middle| x_{t-1}^{(i)}, u_{t-1}\right)}{\pi(x_t^{(i)} | x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t-1})} \cdot \underbrace{\frac{p\left(x_{1:t-1}^{(i)} \middle| z_{1:t-1}, u_{1:t-2}\right)}{\pi\left(x_{1:t-1}^{(i)} \middle| z_{1:t-1}, u_{1:t-2}\right)}}_{w_{t-1}^{(i)}}$$

$$\propto \frac{p\left(z_t \middle| m_{t-1}, x_t^{(i)}\right) p\left(x_t^{(i)} \middle| x_{t-1}^{(i)}, u_{t-1}\right)}{\pi\left(x_t \middle| x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t-1}\right)} \cdot w_{t-1}^{(i)}$$

, in which $\eta = \dfrac{1}{p(z_t | z_{1:t-1}, u_{1:t-1})}$



Sequential Importance Sampling (SIS)

- Now, we select the distribution of last timestep as the source distribution:

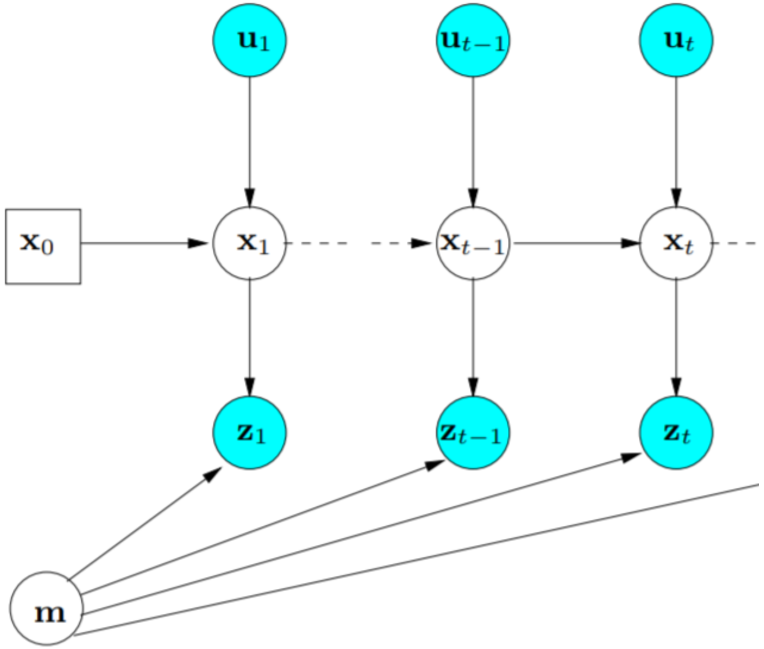
$$\pi\left(x_t \middle| x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t-1}\right) = p(x_t^{(i)} | x_{t-1}^{(i)}, u_{t-1})$$

$$\boxed{w_t^{(i)} = \frac{\eta p\left(z_t \middle| m_{t-1}, x_t^{(i)}\right) p\left(x_t^{(i)} \middle| x_{t-1}^{(i)}, u_{t-1}\right)}{\pi\left(x_t \middle| x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t-1}\right)} \cdot w_{t-1}^{(i)}}$$

- We can get the update weighting:

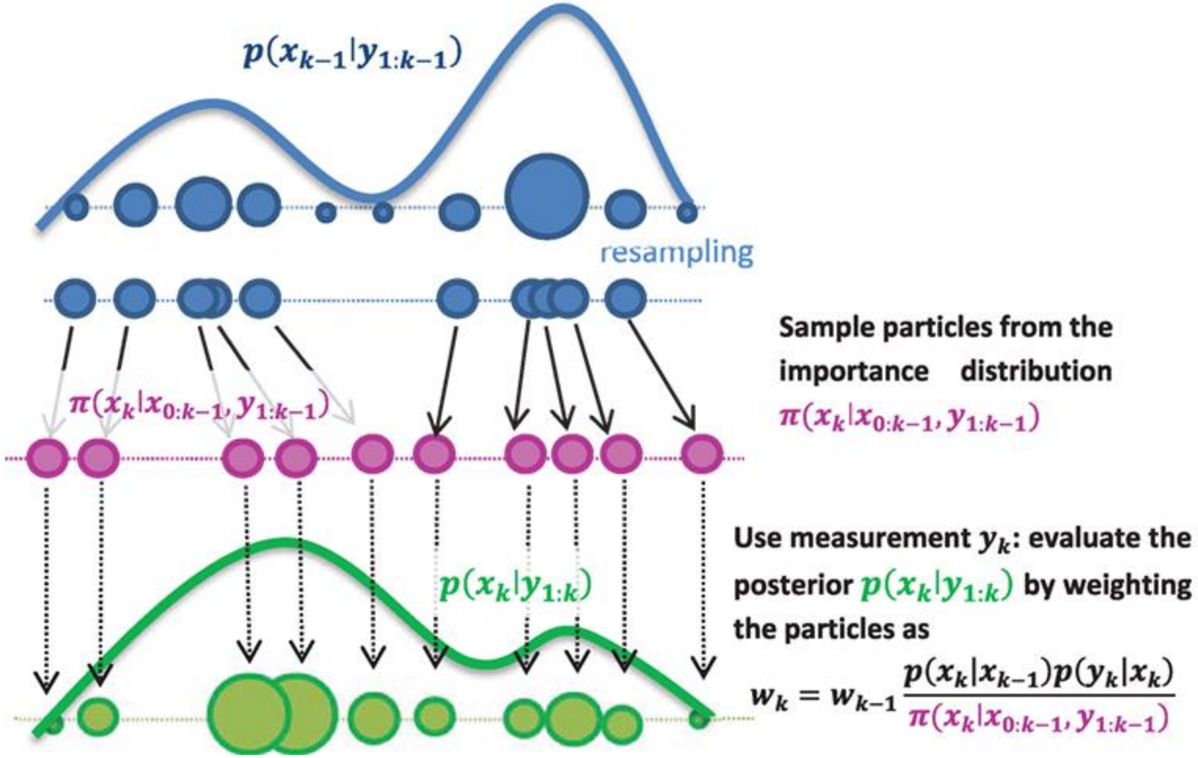
$$w_t^{(i)} = w_{t-1}^{(i)} \frac{\eta p\left(z_t \middle| m_{t-1}, x_t^{(i)}\right) p\left(x_t^{(i)} \middle| x_{t-1}^{(i)}, u_{t-1}\right)}{p(x_t^{(i)} | x_{t-1}^{(i)}, u_{t-1})}$$

$$\propto w_{t-1}^{(i)} \cdot p(z_t | m_{t-1}, x_t^{(i)})$$

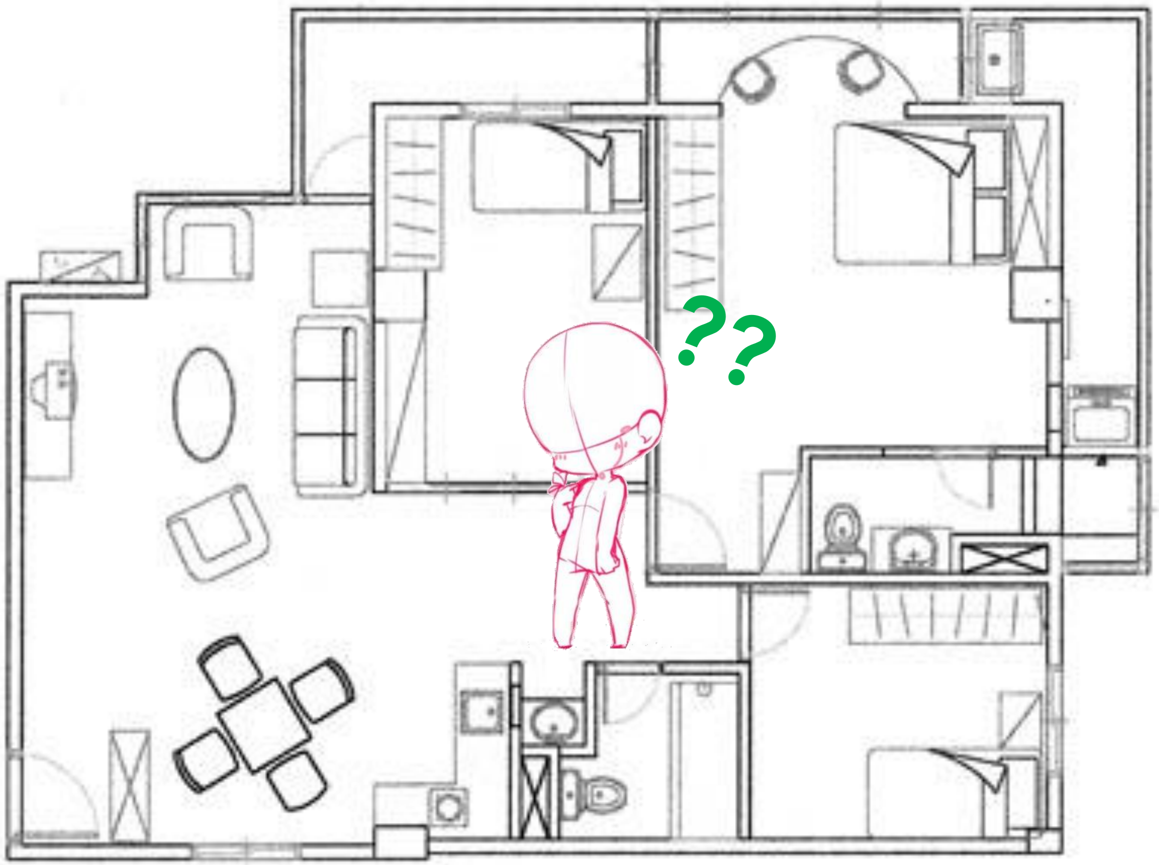


Sequential Importance Resampling (SIR)

- After several steps, the weightings of most particles in SIS particle filter will decrease to close to zero.
- To avoid this problem, we can utilize the resampling process:



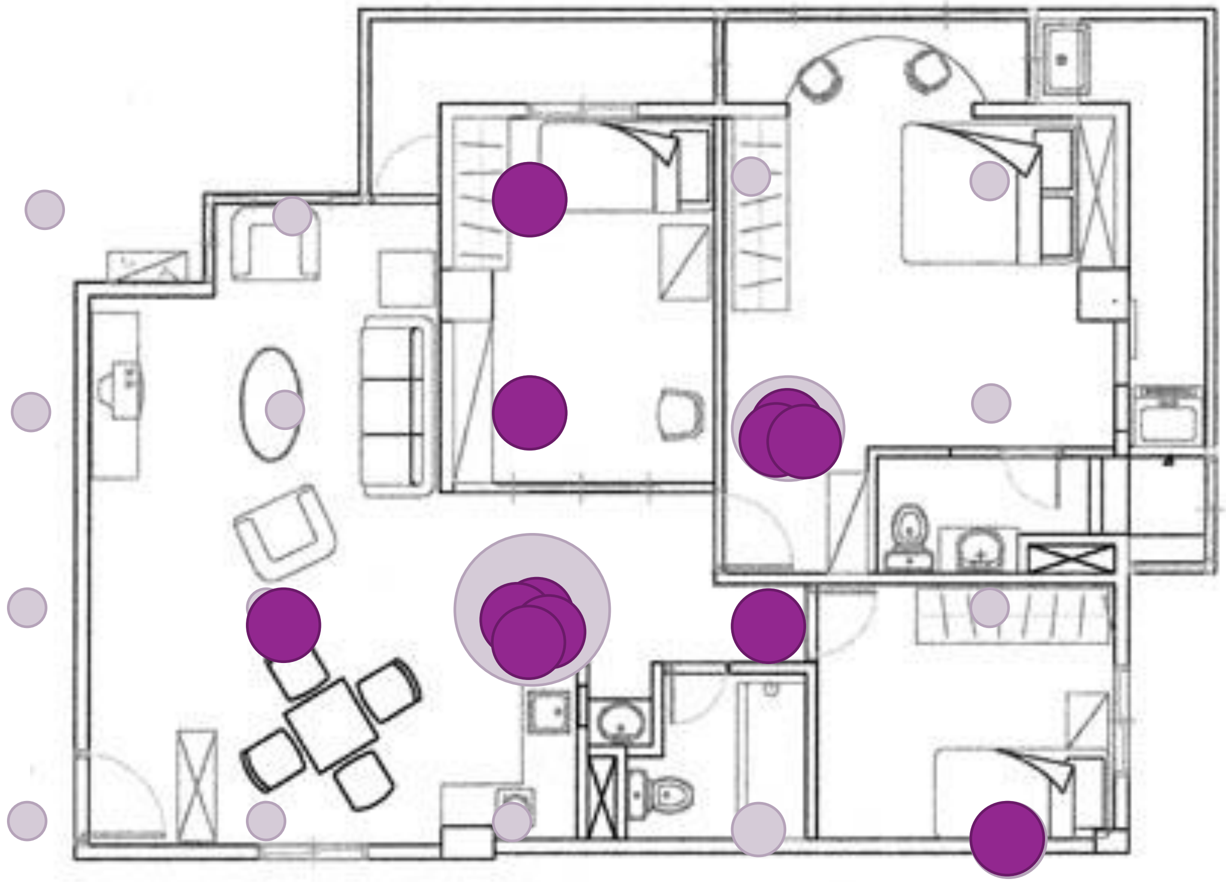
Monte-Carlo Localization Example



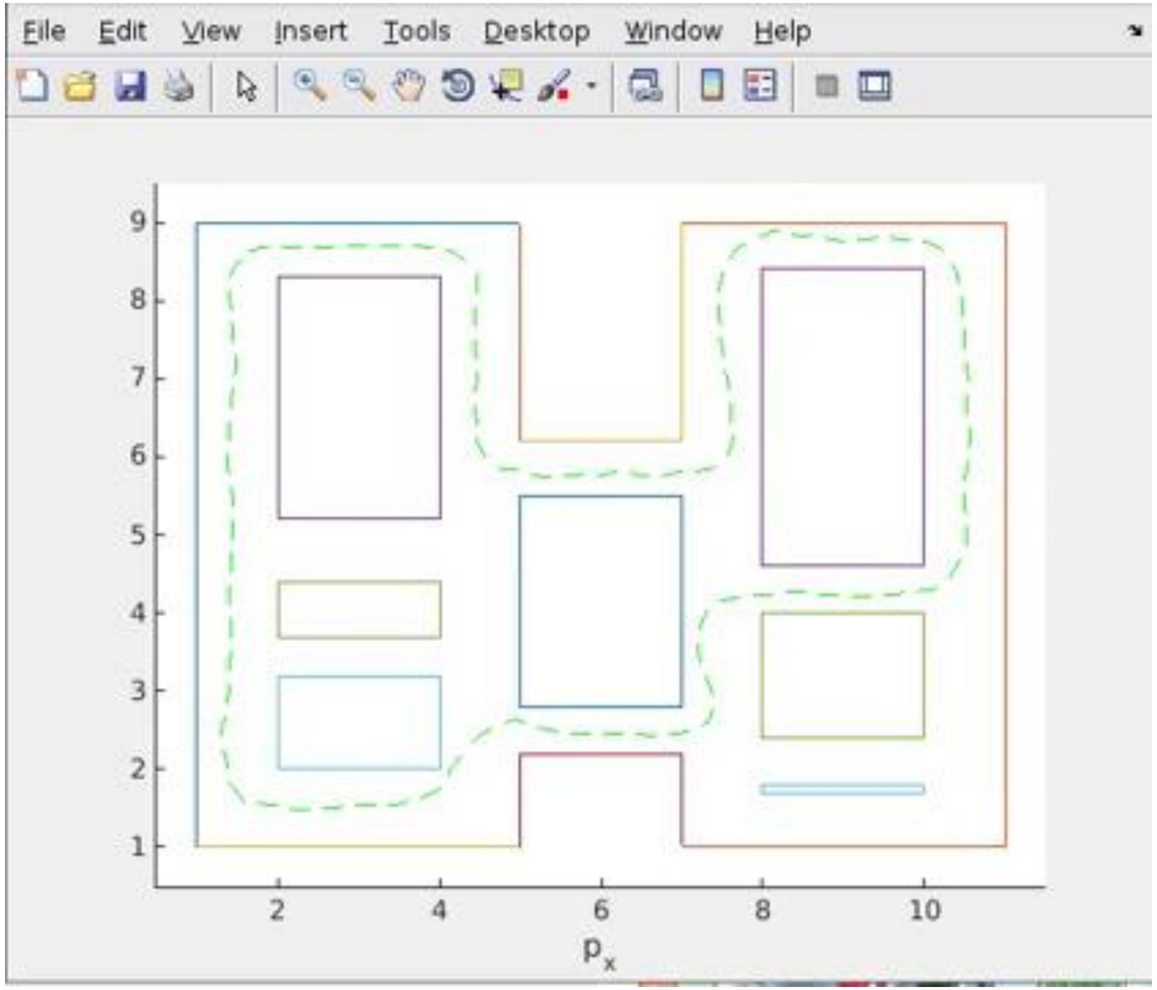
Monte-Carlo Localization Example

Monte-Carlo Localization Example

Monte-Carlo Localization Example



Monte-Carlo Localization



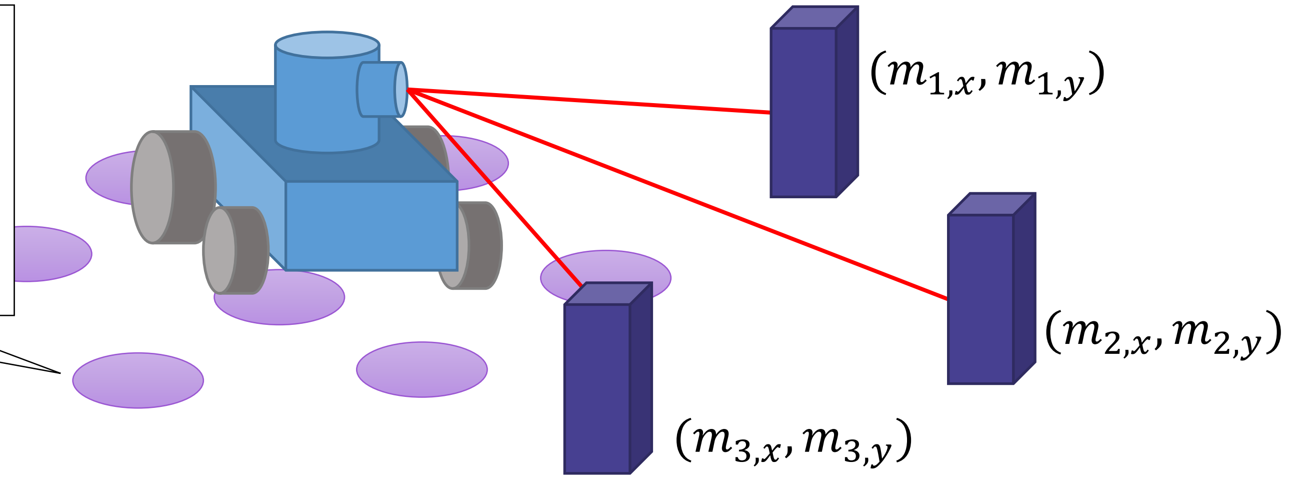
Fast-SLAM

- Now consider the full SLAM problem (localization and mapping), we can divide the full process to localization and mapping steps. This method is called Rao-Blackwellization.

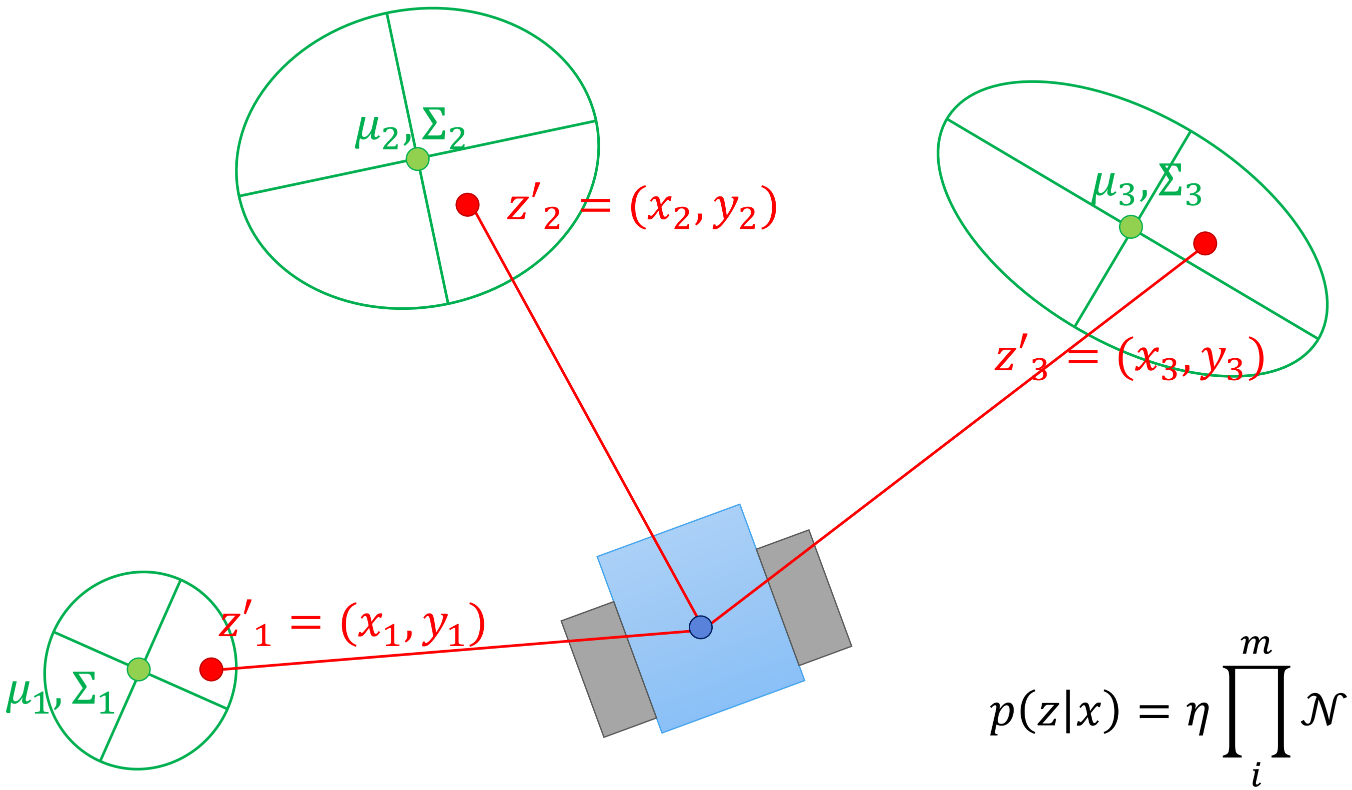
$$p(x_{1:t}, m_t | z_{1:t}, u_{1:t}) = p(x_{1:t} | z_{1:t}, u_{1:t}) p(m_t | x_{1:t}, z_{1:t})$$

- In Fast-SLAM, the robot pose is represented by the multivariate distribution of several weighted particles, and each particle adopts **K** extended Kalman filter to estimate the landmarks independently.

Particle Weights: $w^{(i)}$
Robot Pose: $(x\ y\ \theta)^{(i)}$
Landmarks:
$\left(\mu_1^{(i)}, \Sigma_1^{(i)}\right), \left(\mu_2^{(i)}, \Sigma_2^{(i)}\right), \left(\mu_3^{(i)}, \Sigma_3^{(i)}\right)$



Likelihood of Measurement



$$p(z|x) = \eta \prod_{i}^{m} \mathcal{N}(z'_i; \mu_i, \Sigma_i)$$

Fast-SLAM

- Steps of Fast-SLAM

1. Predict the next pose $x_t^{(i)}$ by motion model.

$$x_t^{(i)} \sim p(x_t^{(i)} | x_{t-1}^{(i)}, u_{t-1})$$

2. Update the distribution of each landmark $(\mu_{j,t}^{(i)}, \Sigma_{j,t}^{(i)})$ via measurement z_k.

$$Q = H\Sigma_{j,t-1}^{(i)}H^T + R, \qquad K_t = \Sigma_{j,t-1}^{(i)}H^T Q^{-1}$$

$$\mu_{j,t}^{(i)} = \mu_{j,t-1}^{(i)} + K_k\left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)})\right)$$

$$\Sigma_{j,t}^{(i)} = (I - K_t H)\Sigma_{j,t-1}^{(i)}$$

3. Update the importance weight of particles.

$$w^{(i)} \sim |2\pi Q|^{-\frac{1}{2}} \exp\{-\frac{1}{2}\left(z_k - h\left(\mu_{j,t-1}^{(i)}, x_t^{(i)}\right)\right)^T Q^{-1}\left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)})\right)\}$$

4. Resampling.

Fast SLAM

- Measure of how well the target distribution is approximated by samples drawn from the proposal.

$$N_{eff} = \frac{1}{\sum_i \left(w_t^{(i)}\right)^2}$$

- N_{eff} denotes the inverse variance of the normalized particle weights. For equal weights, the results is the number of the particles. And the sample approximation is close to the target.

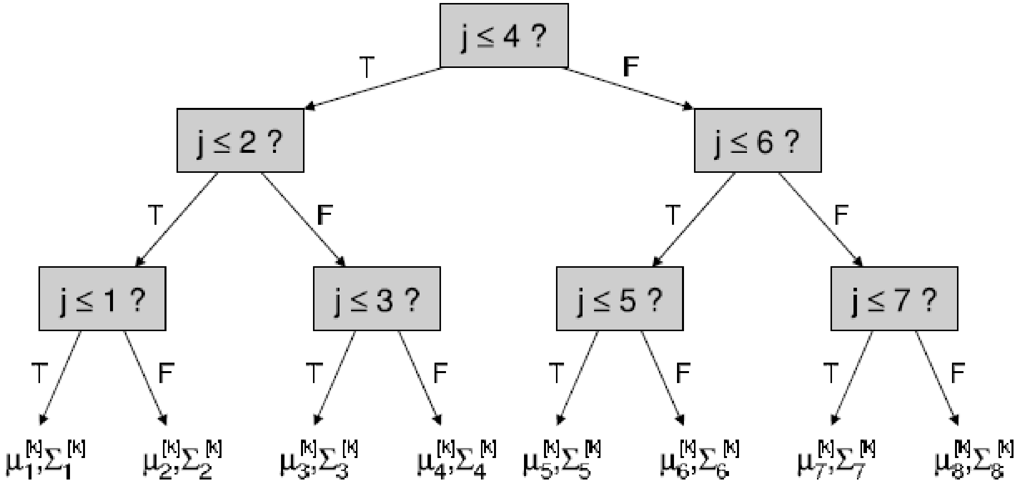
$$N_{eff}^* = \frac{1}{\sum_i \frac{1}{N^2}} = \frac{1}{N \frac{1}{N^2}} = N$$

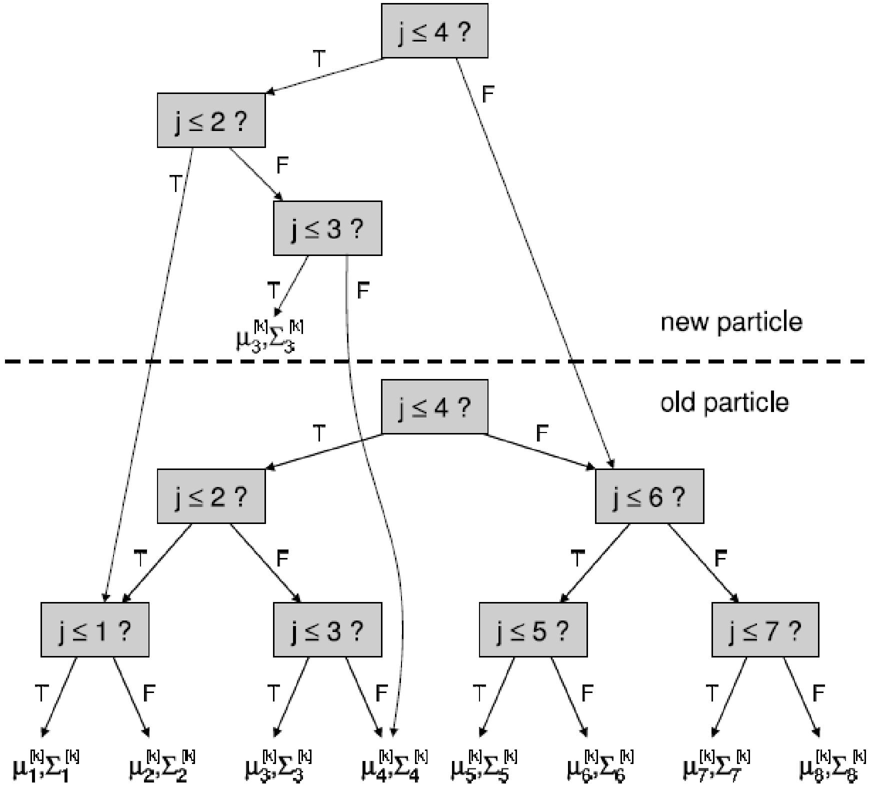
- If N_{eff} drops below a given threshold (usually set to half of the particles), we will resample the particle.

$$N_{eff} < \frac{N}{2}$$

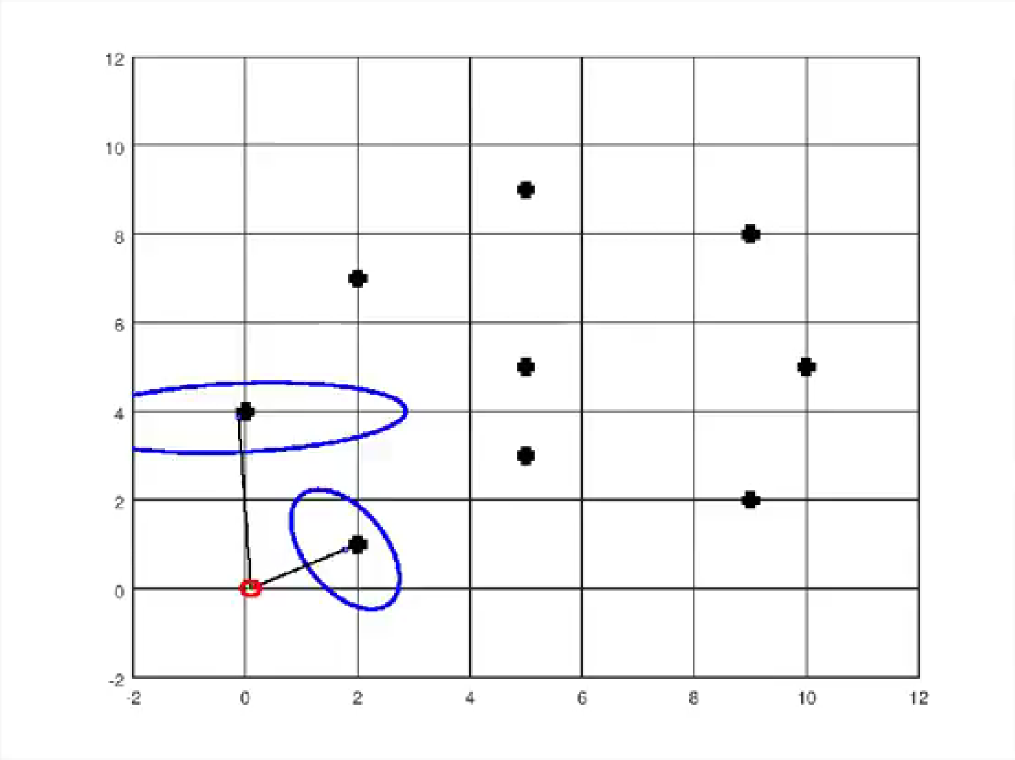
Fast-SLAM

- Efficient implementation of Fast-SLAM. The basic idea is that the set of Gaussians in each particle is represented by a balanced binary tree.





Fast-SLAM Demo



Occupancy Grid Representation

- Occupancy grid maps is a volumetric representation of environments. The advantage of grid representation is that it does not require any predefined definition of landmarks. Instead, in can model arbitrary types of environments.

Grid-based Fast-SLAM

- Steps of Grid-based Fast-SLAM

1. Predict the next pose $x_t^{(i)}$ by motion model.
2. Update the occupancy grid map of each particle.
3. Update the importance weight of particles.
4. Resampling.

Grid-based Fast-SLAM

- Steps of Grid-based Fast-SLAM

1. Predict the next pose $x_t^{(i)}$ by motion model.
2. Update the occupancy grid map of each particle.
3. Update the importance weight of particles.
4. Resampling.

Occupancy Grid Map Algorithm

- The occupancy grids store the probability if the discrete location is free. The state of the grid is defined by the rate of free and occupied.

$$Odd(s) = \frac{p(s=1)}{p(s=0)}$$

- Apply the Bayes theorem to compute the posterior of the state

$$p(s|z) = \frac{p(z|s)p(s)}{p(z)} \qquad Odd(s|z) = \frac{p(s=1|z)}{p(s=0|z)} = \frac{p(z|s=1)p(s=1)/p(z)}{p(z|s=0)p(s=0)/p(z)} = \frac{p(z|s=1)}{p(z|s=0)} Odd(s)$$

- Utilize the log operation to simplify the computation

$$\log Odd(s|z) = \log \frac{p(z|s=1)}{p(z|s=0)} + \log Odd(s)$$

- Compute the occupied probability from log state

define $g = \log Odd(s)$ and $p = p(s=1)$

$$\exp(g) = Odd(s) = \frac{p(s=1)}{p(s=0)} = \frac{p}{1-p} \qquad p = \frac{\exp(g)}{1+\exp(g)}$$

Occupancy Grid Map Algorithm

- Define two likelihood parameter

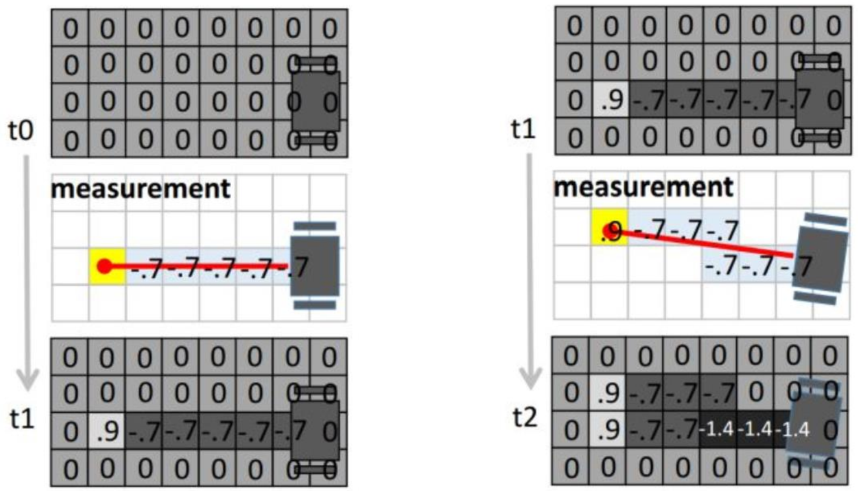
$$l_{occ} = \frac{p(z=0|s=1)}{p(z=0|s=0)} \qquad l_{free} = \frac{p(z=1|s=1)}{p(z=1|s=0)}$$

$$\boxed{\log Odd(s|z) = \log \frac{p(z|s=1)}{p(z|s=0)} + \log Odd(s)}$$

- Initialize the state with half probability of occupied

$$\log Odd(s_{init}) = \log \frac{p(s_{init}=1)}{p(s_{init}=0)} = \log \frac{0.5}{0.5} = 0$$

- Ray tracing the grid, add l_{free} to all the grids passing by, and add l_{occ} to the last grid.

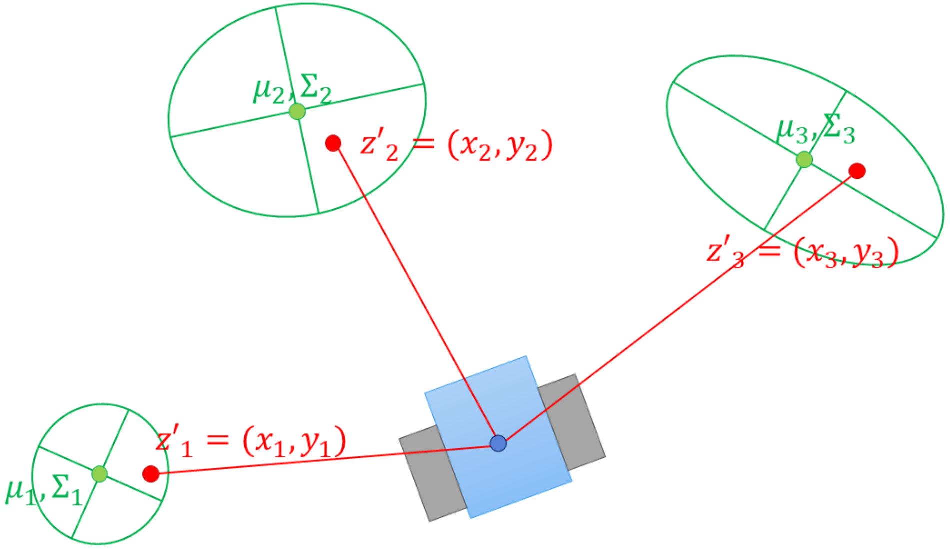


Grid-based Fast-SLAM

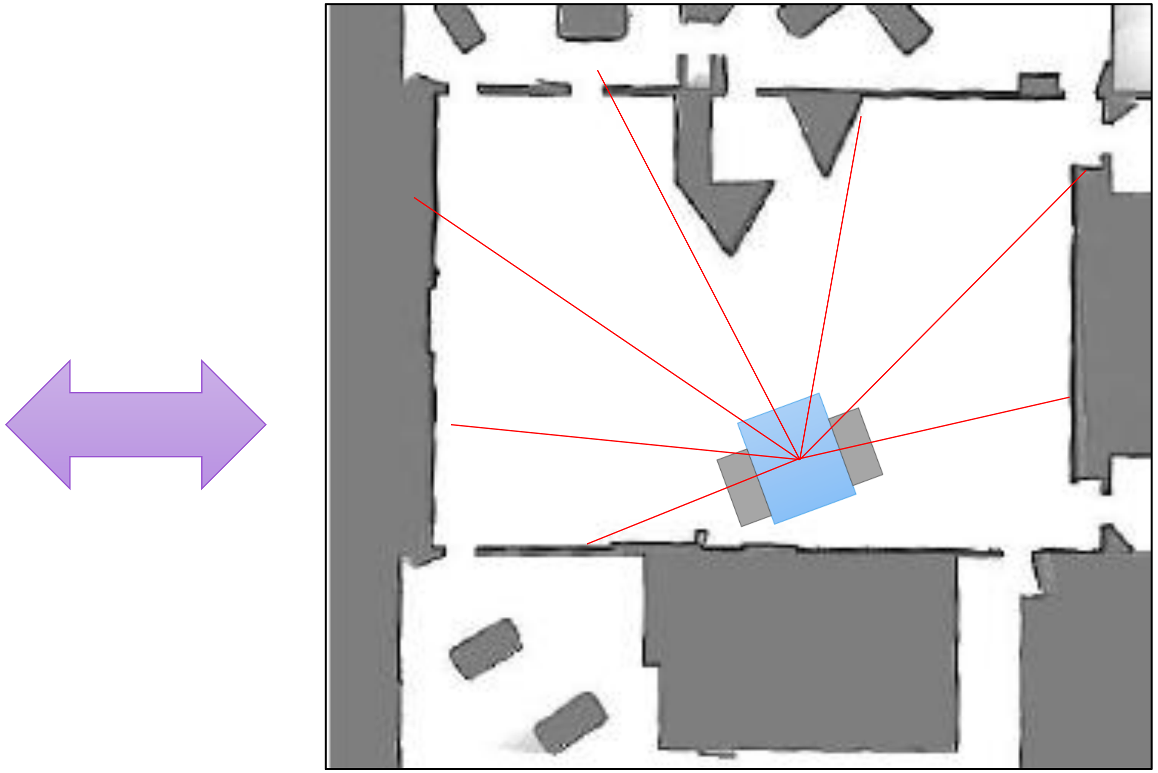
- Steps of Grid-based Fast-SLAM

1. Predict the next pose $x_t^{(i)}$ by motion model.
2. Update the occupancy grid map of each particle.
3. Update the importance weight of particles.
4. Resampling.

Likelihood Field of Grid Map



$$p(z|x) = \eta \prod_{i}^{m} \mathcal{N}(z'_i; \mu_i, \Sigma_i)$$



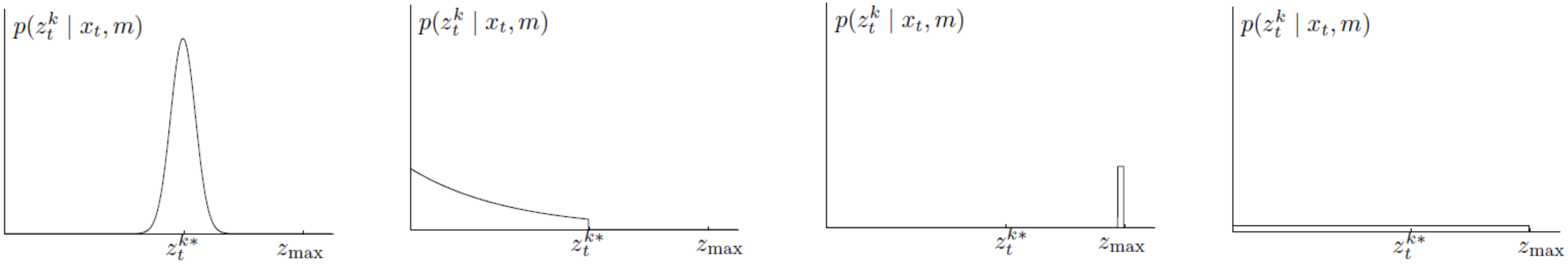
$p(z|x) = ?$

Laser Beam Model

- A common sensor of mobile robot is a range finder, which measures the distance from the robot to the obstacles.

- 4 components of the measurement model:

(a) Gaussian distribution p_{hit} (b) Exponential distribution p_{short} (c) Uniform distribution $p_{\max}$ (d) Uniform distribution p_{rand}



Laser Beam Model

- (a) Correct range with local measurement noise.

$$\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2) = \frac{1}{\sqrt{2\pi\sigma_{\text{hit}}^2}} e^{-\frac{1}{2}\frac{(z_t^k - z_t^{k*})^2}{\sigma_{\text{hit}}^2}}$$

- (b) Unexpected objects

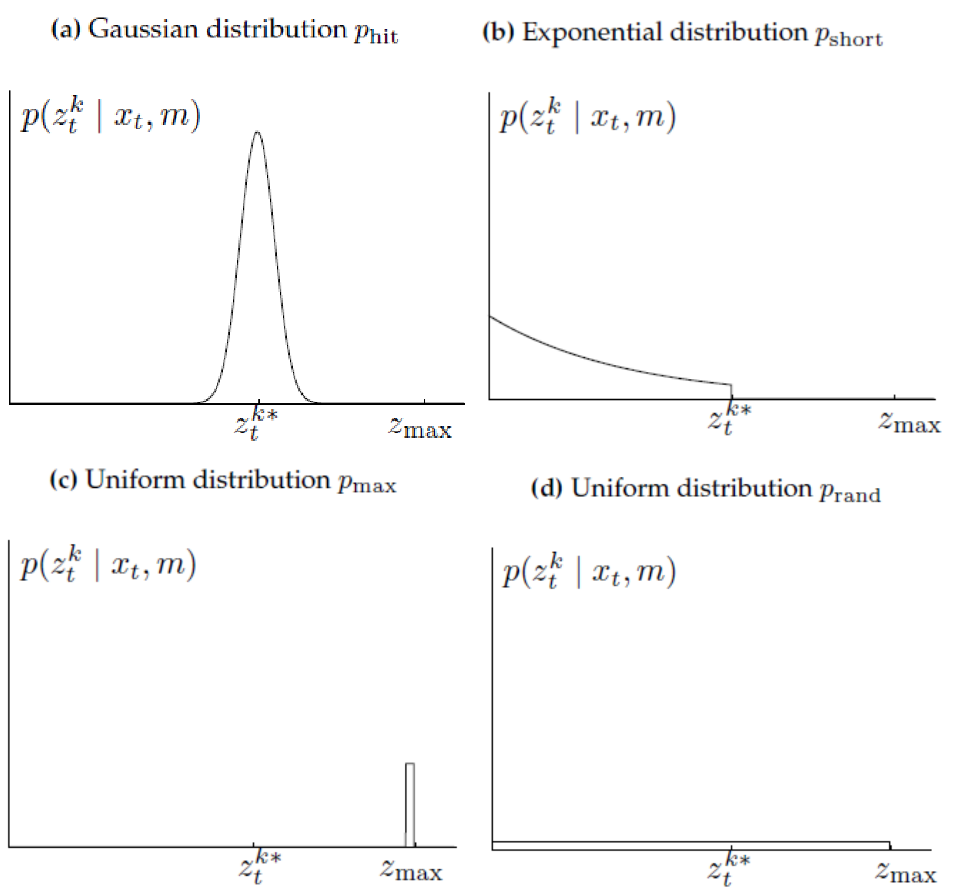
$$p_{\text{short}}(z_t^k \mid x_t, m) = \begin{cases} \eta\, \lambda_{\text{short}}\, e^{-\lambda_{\text{short}} z_t^k} & \text{if } 0 \le z_t^k \le z_t^{k*} \\ 0 & \text{otherwise} \end{cases}$$

- (c) Failures

$$p_{\max}(z_t^k \mid x_t, m) = I(z = z_{\max}) = \begin{cases} 1 & \text{if } z = z_{\max} \\ 0 & \text{otherwise} \end{cases}$$

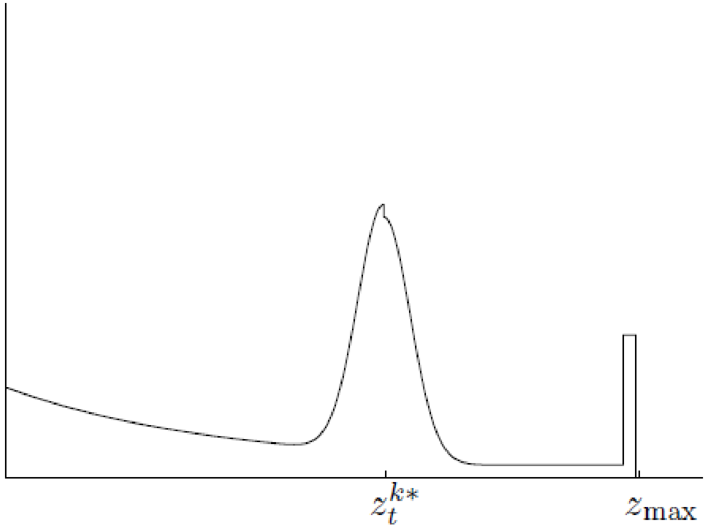
- (d) Random measurements

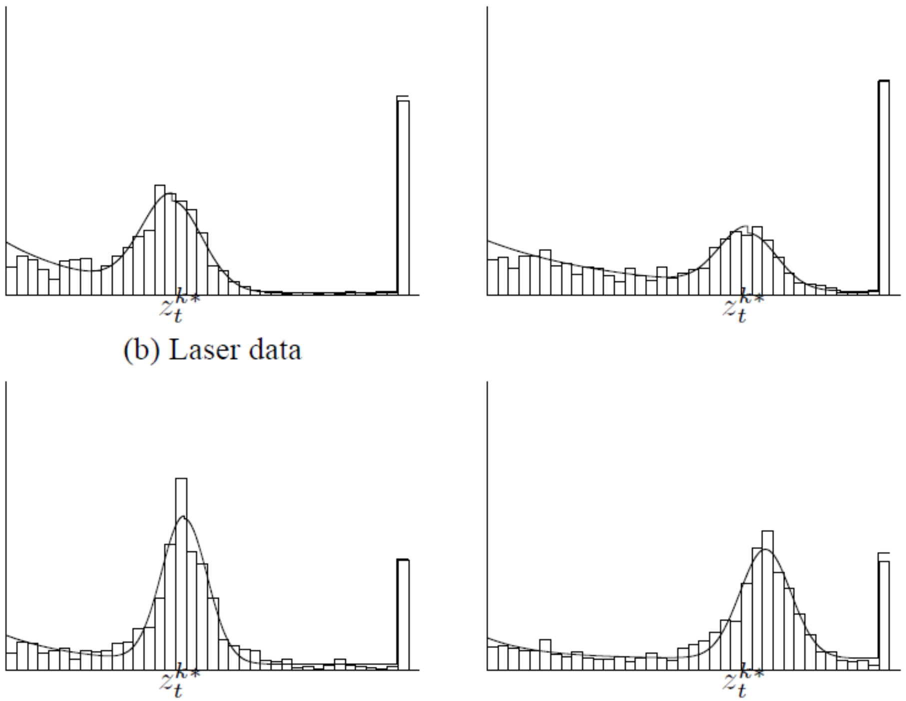
$$p_{\text{rand}}(z_t^k \mid x_t, m) = \begin{cases} \frac{1}{z_{\max}} & \text{if } 0 \le z_t^k < z_{\max} \\ 0 & \text{otherwise} \end{cases}$$



Laser Beam Model

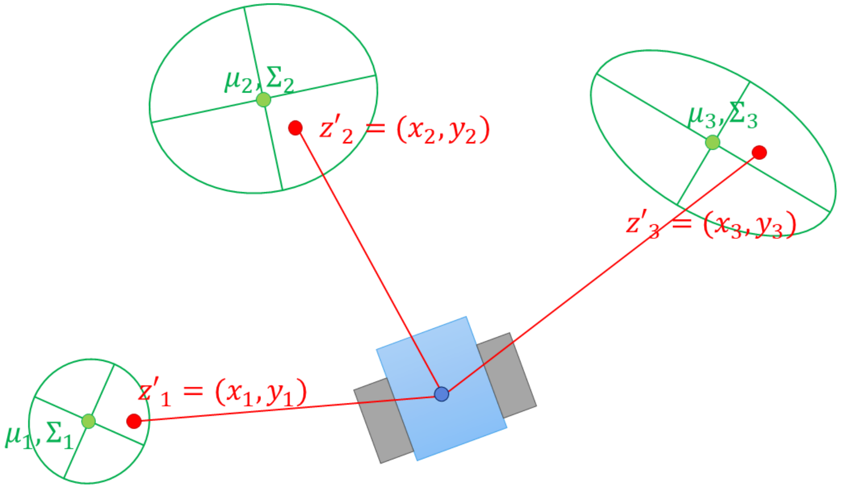
- Mixture distribution of laser beam model.

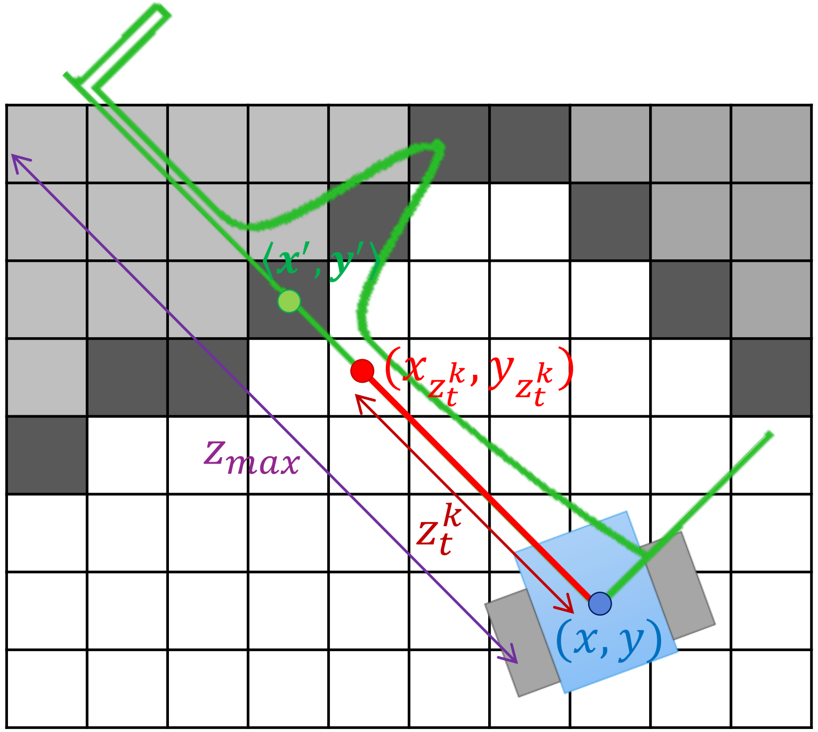




$$p(z_t^k \mid x_t, m) = \begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\text{max}} \\ z_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z_t^k \mid x_t, m) \\ p_{\text{short}}(z_t^k \mid x_t, m) \\ p_{\text{max}}(z_t^k \mid x_t, m) \\ p_{\text{rand}}(z_t^k \mid x_t, m) \end{pmatrix}$$

Likelihood Field





Algorithm likelihood_field_range_finder_model(z_t, x_t, m):

$q = 1$

for all k do

if $z_t^k \neq z_{\max}$

$x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$

$y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$

$dist = \min_{x',y'} \left\{ \sqrt{(x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2} \,\middle|\, \langle x', y' \rangle \text{ occupied in } m \right\}$

$q = q \cdot \left(z_{\text{hit}} \cdot \mathbf{prob}(dist, \sigma_{\text{hit}}) + \frac{z_{\text{random}}}{z_{\max}} \right)$

return q

Grid-based Fast-SLAM

1. Predict the next pose $x_t^{(i)}$ by motion model.

$$x_t^{(i)} \sim p(x_t^{(i)} | x_{t-1}^{(i)}, u_{t-1})$$

2. Update the occupancy grid map of each particle.

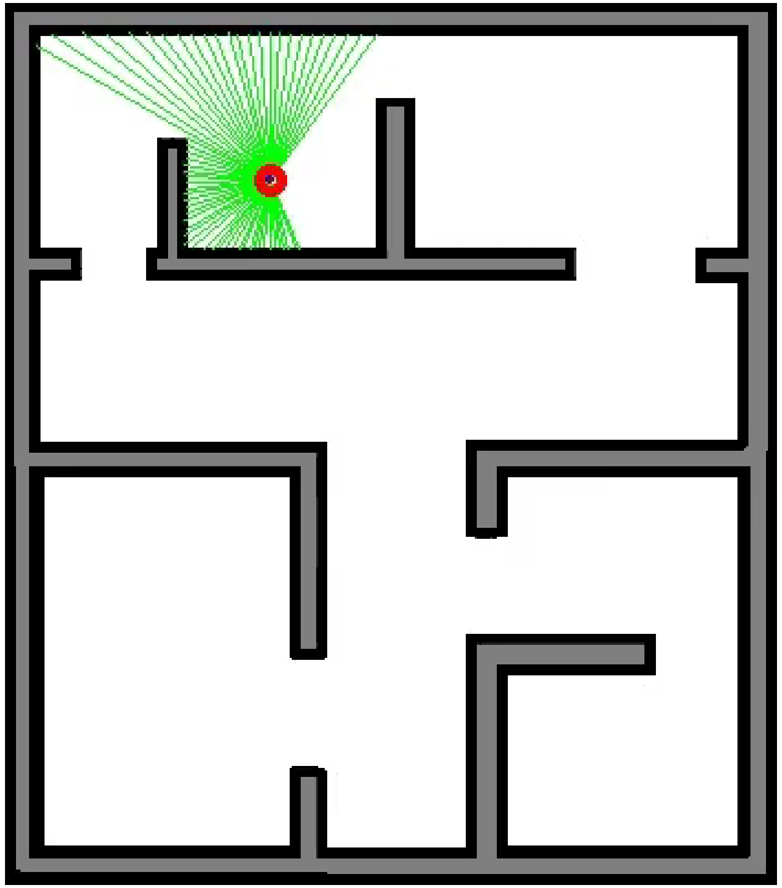
$$\log Odd(s|z) = \log \frac{p(z|s=1)}{p(z|s=0)} + \log Odd(s)$$

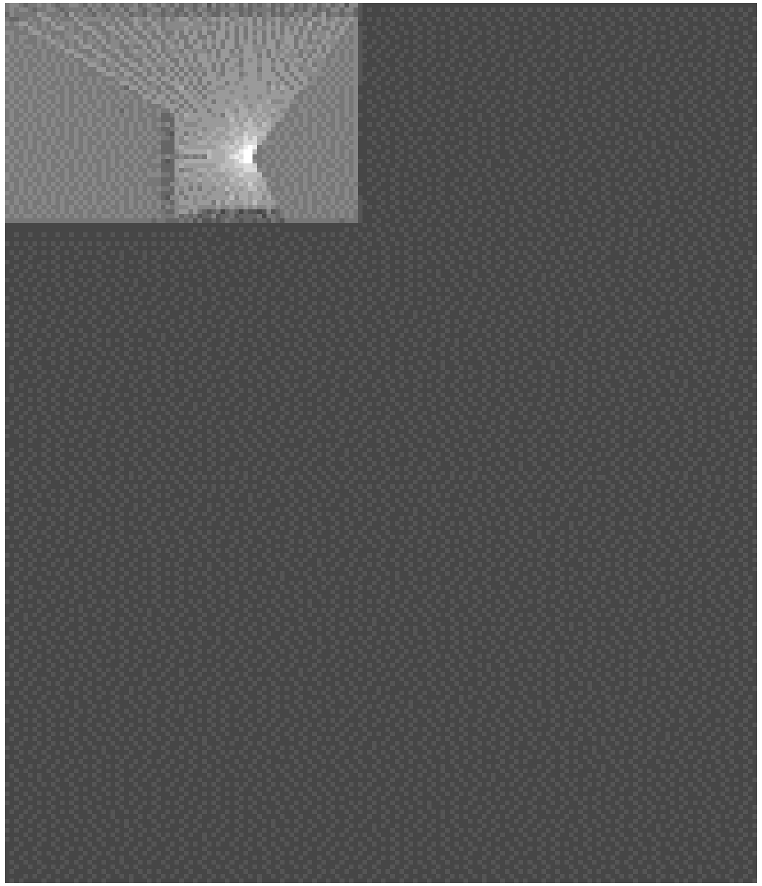
3. Update the importance weight of particles.

$$w_t^{(i)} = \eta \prod_i (z_{hit} \cdot prob(dist, \sigma_{hit}) + \frac{z_{random}}{z_{max}})$$

4. Resampling.

Grid-based Fast-SLAM Demo





Outline

- State Estimation and SLAM Problem
- SLAM Back-end (Error Compensation)
  - Filter-based Methods
    - Probability Theory and Bayes Filter
    - Kalman Filter (KF) / Extended Kalman Filter (EKF)
      - EKF-SLAM
    - Particle Filter
      - Fast-SLAM
  - Graph-based Methods
    - Pose Graph and Least-square Optimization
    - Gauss-Newton and Levenberg-Marquardt Algorithm
    - Sparse Matrix for Optimization

State Estimation

$$\mathbf{x} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \qquad P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N \det(\Sigma)}} \exp(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu}))$$

$$-\ln P(\mathbf{x}) = \frac{1}{2}\ln((2\pi)^N \det(\boldsymbol{\Sigma})) + \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})$$

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k, \quad \mathbf{w}_k \sim \mathcal{N}(0, \mathbf{R}_k)$$

$$\mathbf{z}_{k,j} = h(\mathbf{m}_j, \mathbf{x}_k) + \mathbf{v}_{k,j}, \quad \mathbf{v}_{k,j} \sim \mathcal{N}(0, \mathbf{Q}_{k,j})$$

- Probability of $\{\mathbf{x}_1, \dots, \mathbf{x}_N, \mathbf{m}_1, \dots, \mathbf{m}_M\}$ given $\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_N, \mathbf{z}_{1,1}, \dots, \mathbf{z}_{N,M}\}$:

$$\underbrace{P(\mathbf{x}, \mathbf{m} | \mathbf{z}, \mathbf{u})}_{\text{posterior}} = \frac{P(\mathbf{z}, \mathbf{u} | \mathbf{x}, \mathbf{m}) P(\mathbf{x}, \mathbf{m})}{P(\mathbf{z}, \mathbf{u})} \propto \underbrace{P(\mathbf{z}, \mathbf{u} | \mathbf{x}, \mathbf{m})}_{\text{likelihood}} \underbrace{P(\mathbf{x}, \mathbf{m})}_{\text{prior}}$$

$$(\mathbf{x}, \mathbf{m})^*_{MAP} = \operatorname{argmax} P(\mathbf{x}, \mathbf{m} | \mathbf{z}, \mathbf{u}) = \operatorname{argmax} P(\mathbf{z}, \mathbf{u} | \mathbf{x}, \mathbf{m}) P(\mathbf{x}, \mathbf{m})$$

$$(\mathbf{x}, \mathbf{m})^*_{\mathrm{MLE}} = \operatorname{argmax} P(\mathbf{z}, \mathbf{u} | \mathbf{x}, \mathbf{m})$$

$$P(\mathbf{z}_{k,j} | \mathbf{x}_\mathbf{k}, \mathbf{m}_j) = \mathcal{N}(h(\mathbf{m}_j, \mathbf{x}_k), \mathbf{Q}_{k,j})$$

$$(\mathbf{x}_k, \mathbf{m}_j)^*_{MLE} = \operatorname{argmax} \mathcal{N}(h(\mathbf{m}_j, \mathbf{x}_k), \mathbf{Q}_{k,j}) = \operatorname{argmin} \frac{1}{2}\left(\mathbf{z}_{k,j} - h(\mathbf{m}_j, \mathbf{x}_k)\right)^T \mathbf{Q}_{k,j}{}^{-1} \left(\mathbf{z}_{k,j} - h(\mathbf{m}_j, \mathbf{x}_k)\right)$$

State Estimation

$$(\mathbf{x}_k, \mathbf{m}_j)^*_{MLE} = \operatorname{argmin} \frac{1}{2}\left(\mathbf{z}_{k,j} - h\left(\mathbf{m}_j, \mathbf{x}_k\right)\right)^T \mathbf{Q}_{k,j}{}^{-1} \left(\mathbf{z}_{k,j} - h\left(\mathbf{m}_j, \mathbf{x}_k\right)\right)$$

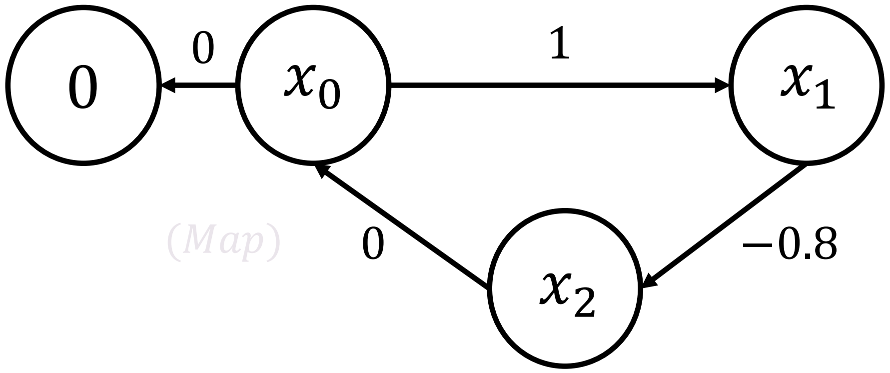
$$\mathrm{P}(\mathbf{z}, \mathbf{u} | \mathbf{x}, \mathbf{m}) = \prod_k P(\mathbf{u}_k | \mathbf{x}_{k-1}, \mathbf{x}_k) \prod_{k,j} P(\mathbf{z}_{k,j} | \mathbf{x}_k, \mathbf{m}_j)$$

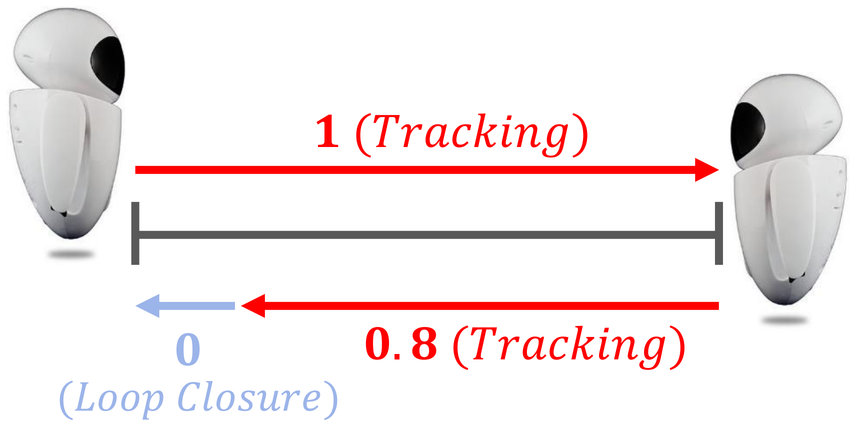
$$\mathbf{e}_{\mathbf{u},k} = \mathbf{x}_k - f(\mathbf{x}_{k-1}, \mathbf{u}_k)$$

$$\mathbf{e}_{\mathbf{z},k,j} = \mathbf{z}_{k,j} - h\left(\mathbf{m}_j, \mathbf{x}_k\right)$$

$$\min F(\mathbf{x}, \mathbf{m}) = \sum_k \mathbf{e}^T_{\mathbf{u},k} \mathbf{R}^{-1}_K \mathbf{e}_{\mathbf{u},k} + \sum_k \sum_j \mathbf{e}^T_{\mathbf{z},k,j} \mathbf{Q}^{-1}_{K,j} \mathbf{e}_{\mathbf{z},k,j}$$

Graph Optimization: 1D Example





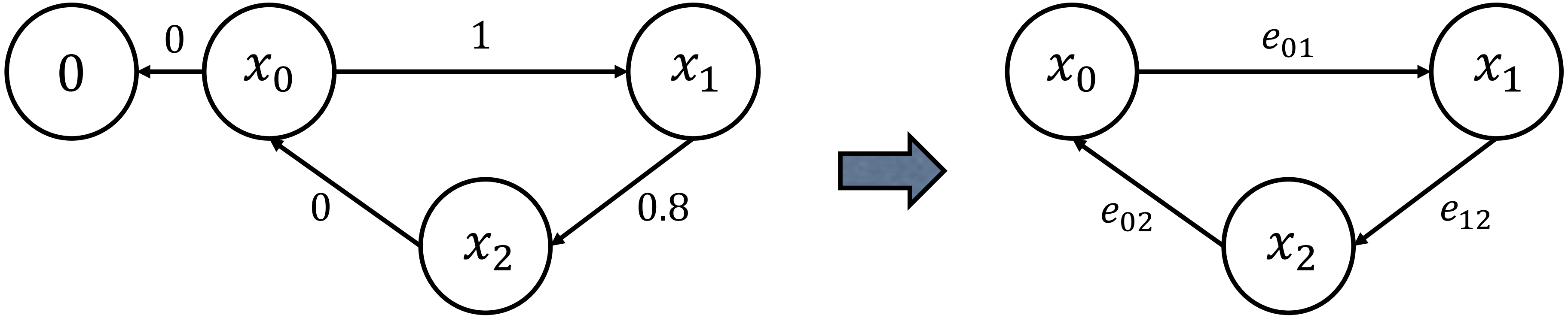
Error function

$$
\begin{aligned}
x_0 &= 0 \\
x_1 &= x_0 + 1 \\
x_2 &= x_1 - 0.8 \\
x_0 &= x_2 + 0
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
f_1 &= x_0 \\
f_2 &= x_1 - x_0 - 1 \\
f_3 &= x_2 - x_1 + 0.8 \\
f_4 &= x_0 - x_2
\end{aligned}
$$

$$
\min_x \sum_i w_i f_i^2 = w_1 x_0^2 + w_2(x_1 - x_0 - 1)^2 + w_3(x_2 - x_1 + 0.8)^2 + w_4(x_0 - x_2)^2
$$

(Optimization)

Graph Optimization: 1D Example



Error Function

$$e_{01} = x_1 - x_0 - 1$$
$$e_{12} = x_2 - x_1 - 0.8$$
$$e_{02} = x_0 - x_2$$

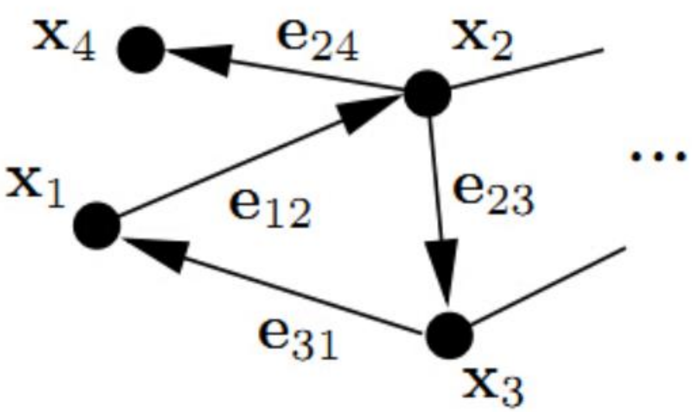
$$\min_{x} \sum_{i,j} w_{ij} e_{ij}^2 = w_{01}(x_1 - x_0 - 1)^2 + w_{12}(x_2 - x_1 + 0.8)^2 + w_{02}(x_0 - x_2)^2$$

Graph Optimization: General Form

$$\min_x \sum_{i,j} w_{ij} e_{ij}^2 = w_{01}(x_1 - x_0 - 1)^2 + w_{12}(x_2 - x_1 + 0.8)^2 + w_{02}(x_0 - x_2)^2$$

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i,j \rangle \in \mathcal{C}} \underbrace{\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})^\top \boldsymbol{\Omega}_{ij} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})}_{\mathbf{F}_{ij}} \quad (1)$$

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \mathbf{F}(\mathbf{x}). \quad (2)$$



$$\begin{aligned} \mathbf{F}(\mathbf{x}) = & \ \mathbf{e}_{12}^\top \boldsymbol{\Omega}_{12} \mathbf{e}_{12} \\ & + \mathbf{e}_{23}^\top \boldsymbol{\Omega}_{23} \mathbf{e}_{23} \\ & + \mathbf{e}_{31}^\top \boldsymbol{\Omega}_{31} \mathbf{e}_{31} \\ & + \mathbf{e}_{24}^\top \boldsymbol{\Omega}_{24} \mathbf{e}_{24} \\ & + \ldots \end{aligned}$$

Graph Optimization Library

g2o - General Graph Optimization

Linux: build passing Windows: build passing

g2o is an open-source C++ framework for optimizing graph-based nonlinear error functions. g2o has been designed to be easily extensible to a wide range of problems and a new problem typically can be specified in a few lines of code. The current implementation provides solutions to several variants of SLAM and BA.

https://github.com/RainerKuemmerle/g2o

Ceres Solver

Ceres Solver is an open source C++ library for modeling and solving large, complicated optimization problems. It is a feature rich, mature and performant library which has been used in production at Google since 2010. Ceres Solver can solve two kinds of problems.

https://github.com/ceres-solver/ceres-solver

Non-linear Optimization

Basics of Optimization

Least Squares Problem

m: number of data points

n: number of parameters

Find $\mathbf{x}^*$, a local minimizer for

$$F(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m}(f_i(\mathbf{x}))^2 ,$$

where $f_i : \mathbb{R}^n \mapsto \mathbb{R}$, $i=1,\ldots,m$ are given functions, and $m \geq n$.

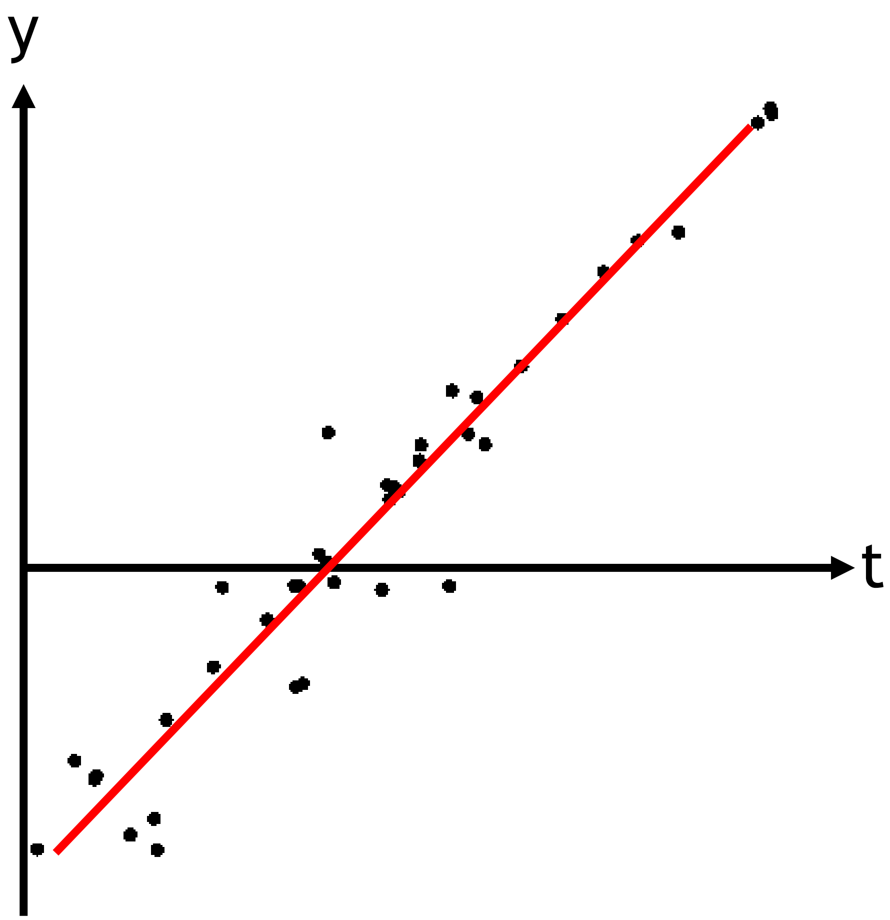
$$\frac{dF}{d\mathbf{x}} = 0$$

Local Minimizer

Given $F : \mathbb{R}^n \mapsto \mathbb{R}$. Find $\mathbf{x}^*$ so that

$$F(\mathbf{x}^*) \leq F(\mathbf{x}) \quad \text{for} \quad \|\mathbf{x} - \mathbf{x}^*\| < \delta .$$

Example: Linear Least Square Fitting



model parameters

$$y(t) = \boxed{M}(t;\boxed{\mathbf{x}}) = x_0 + x_1 t$$

$$f_i(x) = y_i - \boxed{M(t_i;\mathbf{x})}$$

Residual(error) prediction

$M(t;\mathbf{x}) = x_0 + x_1 t + x_2 t^3$ is linear, too.

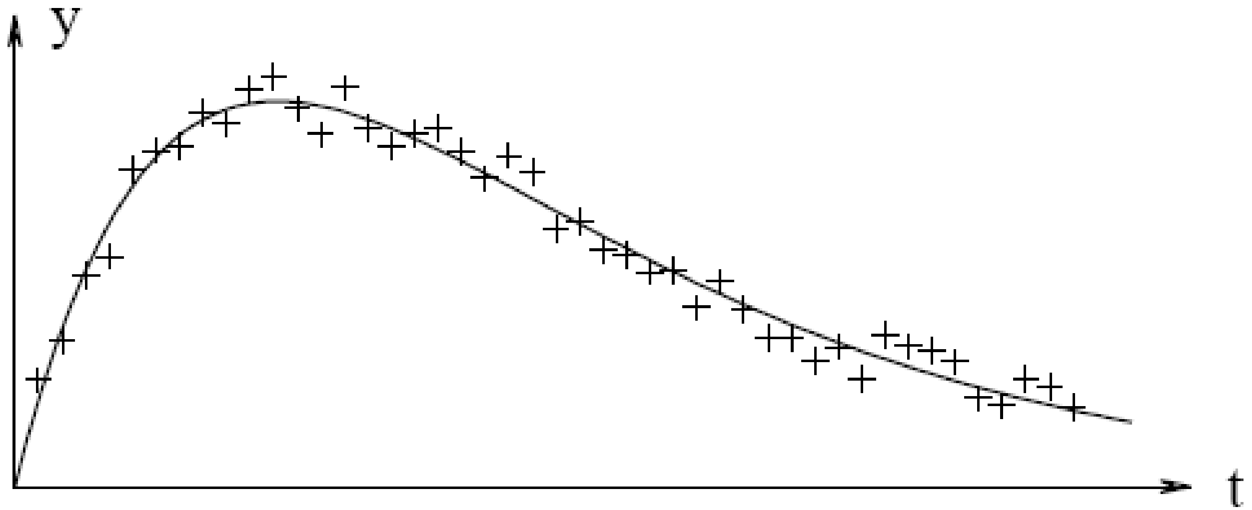
Example: Nonlinear Least Square Fitting

parameters

$$\mathbf{x} = [x_1, x_2, x_3, x_4]^T$$

model

$$M(t;\mathbf{x}) = x_3 e^{x_1 t} + x_4 e^{x_2 t}$$



residuals

$$f_i(\mathbf{x}) = y_i - M(t_i;\mathbf{x})$$
$$= y_i - \left(x_3 e^{x_1 t} + x_4 e^{x_2 t}\right)$$

Function Minimization

Taylor expansion $$F(\mathbf{x}+\mathbf{h}) \approx F(\mathbf{x}) + \boldsymbol{J}(\mathbf{x})^{\boldsymbol{T}}\mathbf{h} + \frac{\mathbf{1}}{\mathbf{2}}\mathbf{h}^{\boldsymbol{T}}\boldsymbol{H}(\mathbf{x})\mathbf{h}$$

$$\boldsymbol{J}(\mathbf{x}) \equiv \boldsymbol{F}'(\mathbf{x}) = \begin{bmatrix} \frac{\partial F}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial F}{\partial x_n}(\mathbf{x}) \end{bmatrix}$$

$$\boldsymbol{H}(\mathbf{x}) \equiv \boldsymbol{F}''(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 F}{\partial x_1^2}(\mathbf{x}) & \frac{\partial^2 F}{\partial x_1 \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \frac{\partial^2 F}{\partial x_2 \partial x_1}(\mathbf{x}) & \frac{\partial^2 F}{\partial x_2^2}(\mathbf{x}) & \cdots & \frac{\partial^2 F}{\partial x_2 \partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial x_n \partial x_1}(\mathbf{x}) & \frac{\partial^2 F}{\partial x_n \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 F}{\partial x_n^2}(\mathbf{x}) \end{bmatrix}$$

Function Minimization

Necessary condition for a local minimizer :

$$\boldsymbol{J}(\mathbf{x}^*) \equiv F'(\mathbf{x}) = \mathbf{0}$$

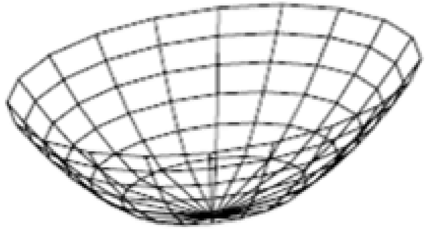
Why?

By definition, if $\mathbf{x}^*$ is a local minimizer,

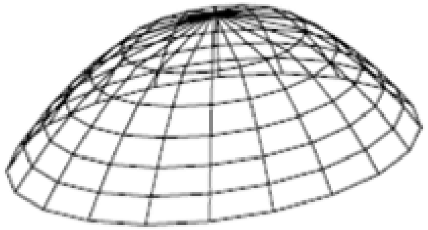
$||\mathbf{h}||$ is small enough → $F(||\mathbf{x}^* + \mathbf{h}||) > F(\mathbf{x}^*)$

$$F(\mathbf{x}^* + \mathbf{h}) \approx F(\mathbf{x}^*) + \boldsymbol{J}(\mathbf{x}^*)^{\boldsymbol{T}}\mathbf{h} + \frac{\mathbf{1}}{\mathbf{2}}\mathbf{h}^{\boldsymbol{T}}\boldsymbol{H}(\mathbf{x})\mathbf{h} > F(\mathbf{x}^*)$$

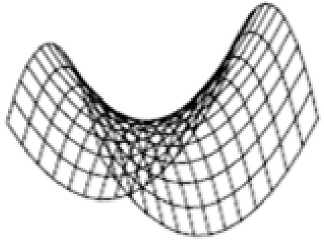
$$F(\mathbf{x}^* - \mathbf{h}) \approx F(\mathbf{x}^*) - \boldsymbol{J}(\mathbf{x}^*)^{\boldsymbol{T}}\mathbf{h} + \frac{\mathbf{1}}{\mathbf{2}}\mathbf{h}^{\boldsymbol{T}}\boldsymbol{H}(\mathbf{x})\mathbf{h} < F(\mathbf{x}^*)$$



a) minimum



b) maximum



c) saddle point

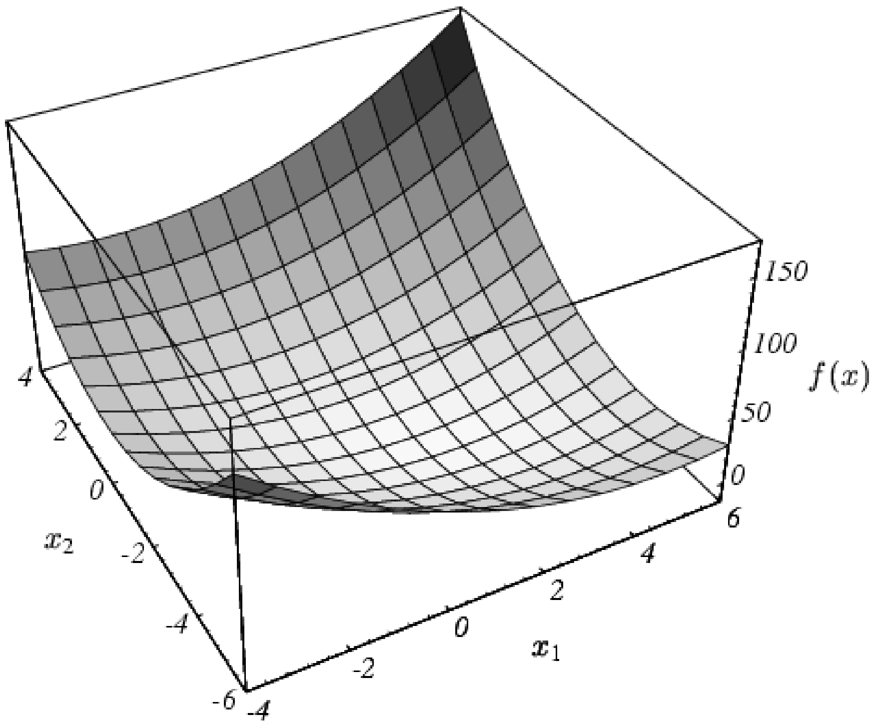
Quadratic Functions

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathbf{T}}\mathbf{A}\mathbf{x} - \mathbf{b}^{\mathbf{T}}\mathbf{x} + \mathbf{c}$$

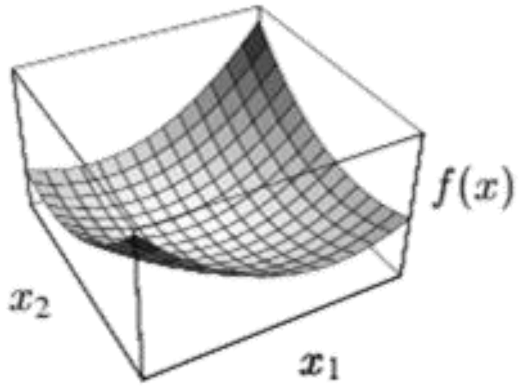
$$\boldsymbol{A} = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$$

$$\boldsymbol{b} = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$

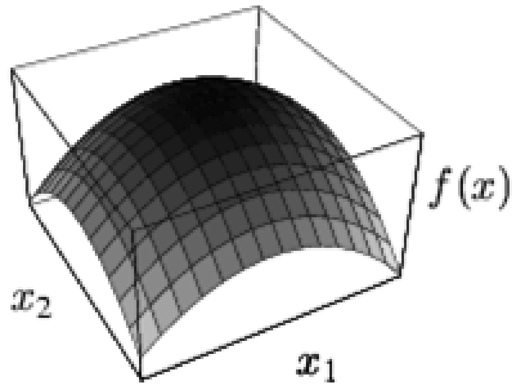
$$\boldsymbol{c} = 0$$



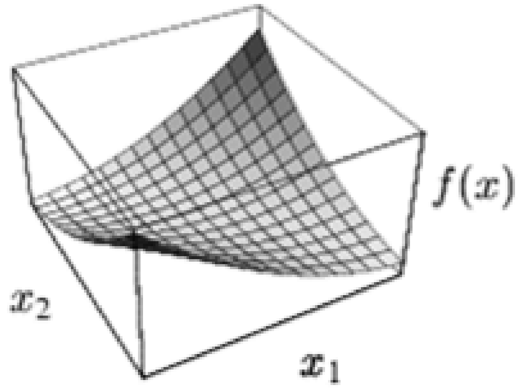
Quadratic Functions



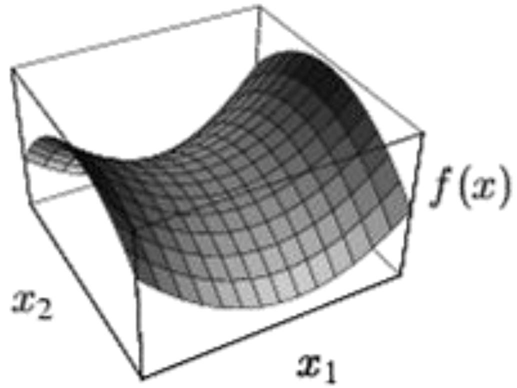
A is positive definite.
All eigenvalues are positive.
For all x, $x^TAx>0$.



A is negative definite.
All eigenvalues are negative.
For all x, $x^TAx<0$.



A is singular



A is indefinite

Descent Methods

$$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k \to \mathbf{x}^* \quad \text{for} \quad k \to \infty$$

1. Find a descent direction $\mathbf{h}_\mathrm{d}$
2. find a step length giving a good decrease in the F-value.

Algorithm Descent method

begin
  $k := 0$; $\mathbf{x} := \mathbf{x}_0$; *found* := **false** {Starting point}
  while (**not** *found*) **and** $(k < k_{\max})$
    $\mathbf{h}_\mathrm{d} :=$ search_direction($\mathbf{x}$) {From $\mathbf{x}$ and downhill}
    if (no such $\mathbf{h}$ exists)
      found := **true** {$\mathbf{x}$ is stationary}
    else
      $\alpha :=$ step_length($\mathbf{x}, \mathbf{h}_\mathrm{d}$) {from $\mathbf{x}$ in direction $\mathbf{h}_\mathrm{d}$}
      $\mathbf{x} := \mathbf{x} + \alpha\mathbf{h}_\mathrm{d}$; $k := k+1$ {next iterate}
end

Descent Direction (Line Search Method)

$$
\begin{aligned}
F(\mathbf{x}+\alpha\mathbf{h}) &= F(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) + O(\alpha^2) \\
&\simeq F(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) \quad \text{for } \alpha \text{ sufficiently small.}
\end{aligned}
$$

Definition of descent direction:

$\mathbf{h}$ is a descent direction for F at $\mathbf{x}$ if $\mathbf{hF}'(\mathbf{x}) < 0$

Steepest Descent Method

$$F(\mathbf{x}+\alpha\mathbf{h}) = F(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) + O(\alpha^2)$$
$$\simeq F(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) \quad \text{for } \alpha \text{ sufficiently small.}$$

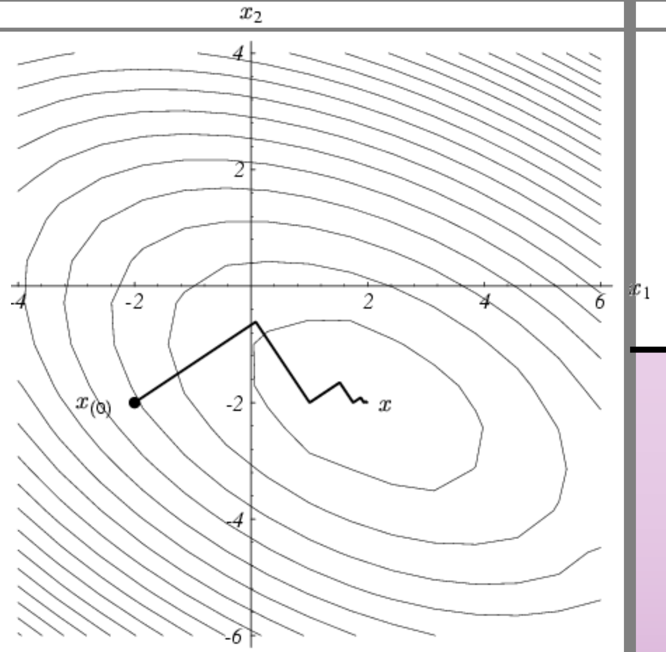
$$\boxed{\frac{F(\mathbf{x}) - F(\mathbf{x}+\alpha\mathbf{h})}{\alpha\|\mathbf{h}\|}} = -\frac{1}{\|\mathbf{h}\|}\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) = -\|\mathbf{F}'(\mathbf{x})\| \cos\theta$$

the decrease of ***F(x)*** per unit along h direction

$$\text{greatest gain rate if } \theta = \pi \quad \rightarrow \quad \mathbf{h}_{\text{sd}} = -\mathbf{F}'(\mathbf{x})$$

h_{sd} is a descent direction because $\mathbf{h}_{sd}^T F'(\mathbf{x}) = -F'(\mathbf{x})^2 < 0$

Steepest Descent Method



$\varphi(\alpha) = F(\mathbf{x} + \alpha\mathbf{h})$, $\mathbf{x}$ and $\mathbf{h}$ are fixed, $\alpha \geq 0$.

Find α so that $\varphi(\alpha) = F(\mathbf{x} + \alpha\mathbf{h})$ is minimum.

$$0 = \frac{\partial\varphi(\alpha)}{\partial\alpha} = \frac{\partial F(\mathbf{x}+\alpha\mathbf{h})}{\partial\alpha} = \frac{\partial F(\mathbf{x}+\alpha\mathbf{h})}{\partial(\mathbf{x}+\alpha\mathbf{h})}\frac{\partial(\mathbf{x}+\alpha\mathbf{h})}{\partial\alpha} = \mathbf{h}^T F'(\mathbf{x} + \alpha\mathbf{h})$$

$$\mathbf{h} = -F'(\mathbf{x})$$

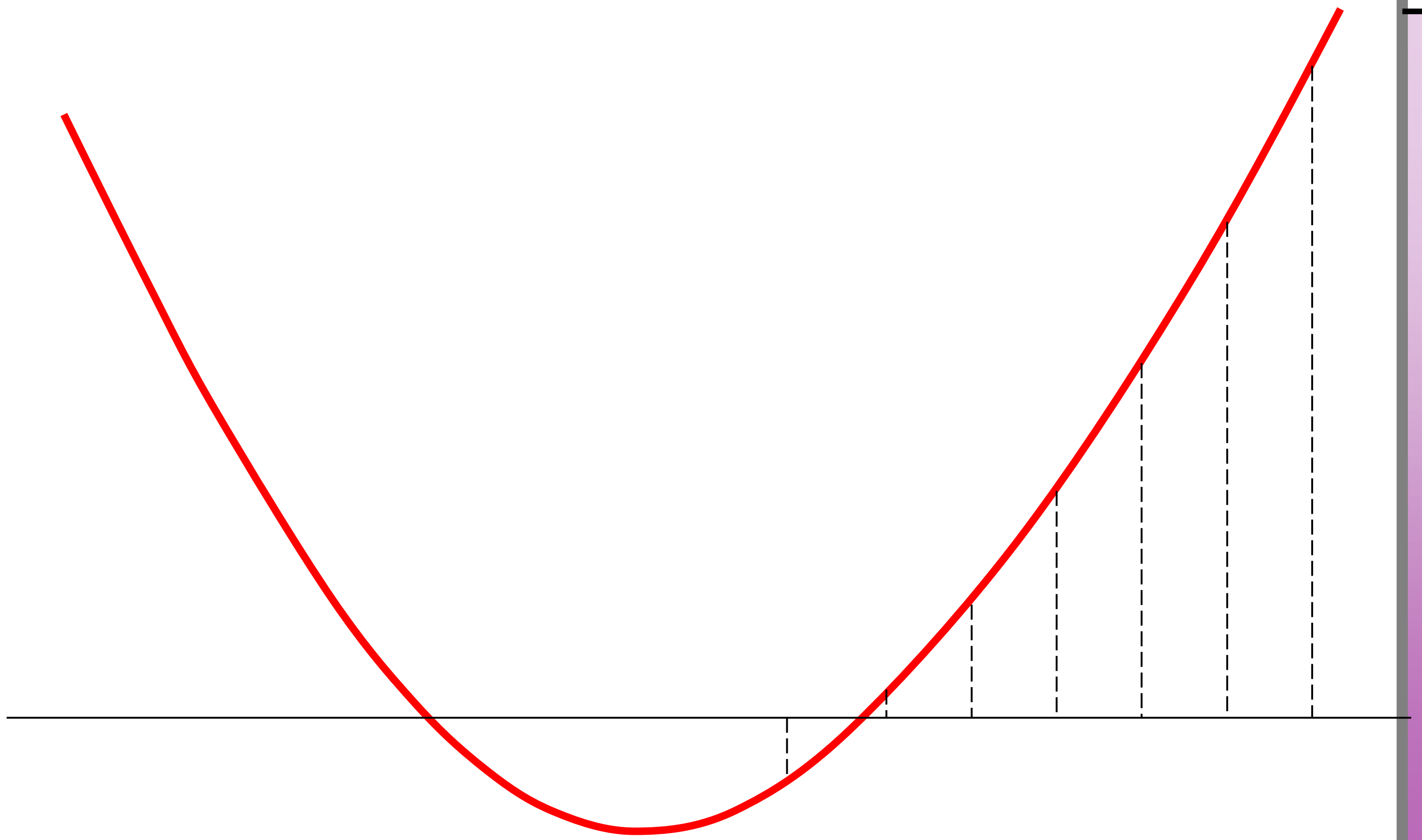
$$=\mathbf{h}^T\ (F'(\mathbf{x}) + \alpha\boldsymbol{F}''(\mathrm{x})^{\boldsymbol{T}}\mathbf{h})=\mathbf{h}^T(-\mathbf{h} + \alpha\mathbf{H}\mathbf{h})$$

$$\alpha = \frac{\mathbf{h}^{\mathrm{T}}\mathbf{h}}{\mathbf{h}^{\mathrm{T}}\mathbf{H}\mathbf{h}}$$

Problem: Has good performance in the initial stages of the iterative process, but converge very slow with a linear rate.

Newton's Method

- Root finding for $f(x)=0$
- March x and test signs
- Determine Δx
  (small→slow; large→ miss)



Newton's Method

- Root finding for $f(x)=0$

Taylor's expansion:

$$f(x_0+\varepsilon) = f(x_0) + f'(x_0)\varepsilon + \frac{1}{2}f''(x_0)\varepsilon^2 + \ldots$$

$$0 = f(x_0+\varepsilon) \approx f(x_0) + f'(x_0)\varepsilon$$

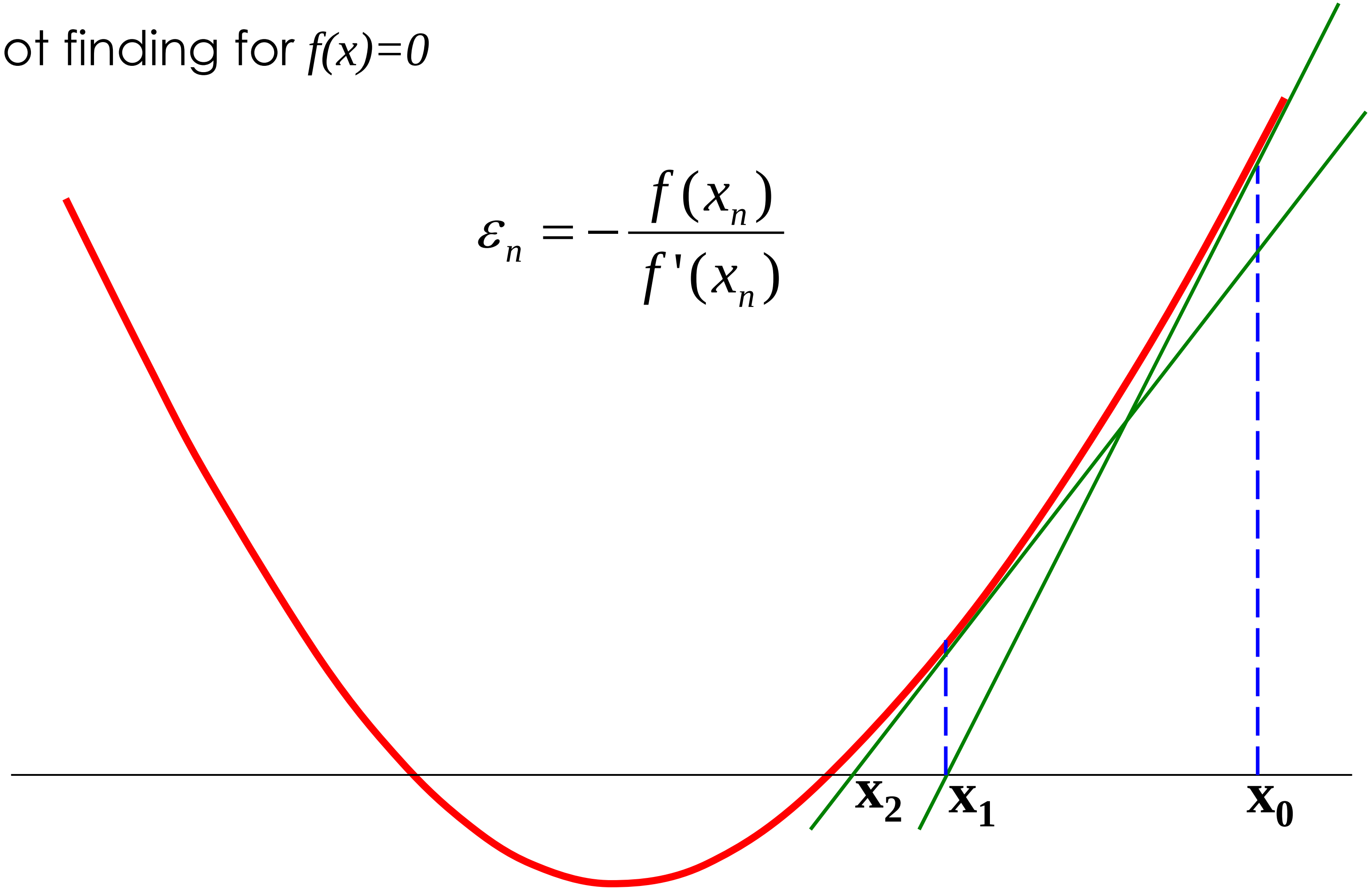
$$\varepsilon = -\frac{f(x_0)}{f'(x_0)}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Newton's Method

- Root finding for $f(x)=0$

$$\varepsilon_n = -\frac{f(x_n)}{f'(x_n)}$$



Newton's Method

$\mathbf{x}^*$ is a stationary point $\longrightarrow$ it satisfies $\mathbf{F}'(\mathbf{x}^*) = \mathbf{0}$.

$$\begin{aligned} \mathbf{F}'(\mathbf{x}+\mathbf{h}) &= \mathbf{F}'(\mathbf{x}) + \mathbf{F}''(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^2) \\ &\simeq \mathbf{F}'(\mathbf{x}) + \mathbf{F}''(\mathbf{x})\mathbf{h} \quad \text{for } \|\mathbf{h}\| \text{ sufficiently small} \\ &= 0 \end{aligned}$$

$\longrightarrow$ $\mathbf{H}\,\mathbf{h}_\mathrm{n} = -\mathbf{F}'(\mathbf{x}) \quad \text{with } \mathbf{H} = \mathbf{F}''(\mathbf{x})$

$\mathbf{x} := \mathbf{x} + \mathbf{h}_\mathrm{n}$

Suppose that $\mathbf{H}$ is positive definite

$\longrightarrow$ $\mathbf{u}^\top \mathbf{H}\,\mathbf{u} > 0$ for all nonzero $\mathbf{u}$.

$\longrightarrow$ $0 < \mathbf{h}_\mathrm{n}^\top \mathbf{H}\,\mathbf{h}_\mathrm{n} = -\mathbf{h}_\mathrm{n}^\top \mathbf{F}'(\mathbf{x})$

$\longrightarrow$ $\mathbf{h}_\mathrm{n}$ is a descent direction

Newton's Method

$$\mathbf{Hh} = -F'(\mathbf{x})$$

$$\mathbf{h} = -\mathbf{H}^{-1}\mathbf{J}$$

- It has good performance in the final stage of the iterative process, where x is close to x*.
- It requires solving a linear system and H is not always positive definite.

➔ Use the approximate Hessian $\mathbf{H} \approx \mathbf{J}^{\mathbf{T}}\mathbf{J}$ Gauss-Newton

Gauss-Newton

$$\mathbf{h}^* = \operatorname{argmin} \frac{1}{2} \sum_{i=1}^{m} \|f_i(\mathbf{x} + \mathbf{h})\|^2$$

$$f(\mathbf{x} + \mathbf{h}) \approx f(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T \mathbf{h}$$

$$\frac{1}{2}\|f(\mathbf{x} + \mathbf{h})\|^2 \approx \frac{1}{2}\left\|f(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T \mathbf{h}\right\|^2 = \frac{1}{2}\left(\|f(\mathbf{x})\|^2 + 2f(\mathbf{x})\mathbf{J}(\mathbf{x})^T\mathbf{h} + \mathbf{h}^T\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^T\mathbf{h}\right)$$

$$\mathbf{J}(\mathbf{x})f(\mathbf{x})^T + \mathbf{J}(\mathbf{x})\mathbf{J}(\mathrm{x})^T\mathbf{h} = \mathbf{0}$$

$$\underbrace{\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^T}_{\mathbf{H}(\mathbf{x})}\mathbf{h} = -\underbrace{\mathbf{J}(\mathbf{x})f(\mathbf{x})^T}_{\mathbf{g}(\mathbf{x})}$$

Newton's Method:

$$\mathbf{Hh} = -F'(\mathbf{x})$$

Levenberg-Marquardt Method (LM)

- LM can be thought of as a combination of steepest descent and the Newton method.
  - When the current solution is far from the correct one, the algorithm behaves like a steepest descent method: slow, but guaranteed to converge.
  - When the current solution is close to the correct solution, it becomes a Newton's method.

$$
\begin{array}{l}
\textbf{if } \mathbf{F}''(\mathbf{x}) \text{ is positive definite} \\
\quad \mathbf{h} := \mathbf{h}_{\mathrm{n}} \\
\textbf{else} \\
\quad \mathbf{h} := \mathbf{h}_{\mathrm{sd}} \\
\mathbf{x} := \mathbf{x} + \alpha \mathbf{h}
\end{array}
$$

true-region method

$$
\boldsymbol{\rho} = \frac{\boldsymbol{f}(\mathbf{x}+\mathbf{h}) - \boldsymbol{f}(\mathbf{x})}{\mathbf{J}(\mathbf{x})^{T}\mathbf{h}}
$$

This needs to calculate second-order derivative which might not be available.

Levenberg-Marquardt Method (LM)

Initialize $\mathbf{x} = \mathbf{x}_0, \mu = \mu_0$

For i=0~K

Find $\mathbf{h}$ such that $\min_{\mathbf{h}} \frac{1}{2}\left\|\mathrm{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T\mathbf{h}\right\|^2$ s.t $\|\mathbf{D}\mathbf{h}\|^2 \leq \mu$

Calculate ρ

If $\rho \geq \frac{\mathbf{3}}{\mathbf{4}}$

$\mu = 2\mu$

If $\rho < \frac{\mathbf{1}}{\mathbf{4}}$

$\mu = 0.5\mu$

If $\rho \geq Th$

else $\mathbf{x} = \mathbf{x} + \mathbf{h}$

If $\mathbf{h}$ is smaller than ϵ, stop

$$\mathcal{L}(\mathbf{h}, \lambda) = \frac{1}{2}\left\|\mathrm{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T\mathbf{h}\right\|^2 + \lambda(\|\mathbf{D}\mathbf{h}\|^2 - \mu)$$

$$\nabla\mathcal{L}(\mathbf{h}, \lambda) = 0$$

$$\left(\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^T + \lambda\mathbf{D}^{\mathbf{T}}\mathbf{D}\right)\mathbf{h} = -\mathbf{J}(\mathbf{x})f(\mathbf{x})^T$$

$$(\mathbf{H}(\mathbf{x}) + \lambda\mathbf{I})\mathbf{h} = -\mathbf{g}(x)$$