



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Course Overview and Introduction to Sensor Fusion

Simo Särkkä

Aalto University

September 11, 2020

Contents

- 1 Course Organization
- 2 Overview of Sensor Fusion
- 3 Example: Sensor Fusion in Drone and Autonomous Car
- 4 Summary

The Team

Simo Särkkä

Lecturer

simo.sarkka@aalto.fi



Muhammad Emzir

Exercises

muhammad.emzir@aalto.fi



Sakira Hassan

Project Work

syeda.s.hassan@aalto.fi



Intended Learning Outcomes of The Course

After successfully completing this course, you are able to:

- explain the principles and components of sensor fusion systems,
- identify and explain the differences between linear and nonlinear models and their implications on sensor fusion,
- construct models of multi-sensor systems and use least-squares algorithms for sensor fusion,
- construct continuous and discrete time state-space models based on ordinary differential equations, difference equations, and physical sensor models,
- develop and compare state-space models and Kalman as well as particle filtering algorithms for solving sensor fusion problems.

Schedule (1)

- 12 virtual lectures
 - Fridays, 12:15 - 14:00 in Zoom, the lectures are recorded.
 - Individual link for each lecture (to ease recording).
 - Detailed *preliminary* schedule (incl. topics) on MyCourses
- 11 virtual exercise sessions
 - Exercise sessions are on Tuesdays, 12:15 - 14:00 in Zoom.
 - Exercises start on Tuesday, September 15, 2020.
- No lectures or exercise sessions on the exam week 19.10. - 23.10.
- Exam on Friday December 11, 2020, 12.00 - 15.00 on MyCourses.

Check MyCourses regularly for updates!

Course Literature

- Lecture notes and slides are the main course literature
 - Lecture notes (~ course book) are already available on the course homepage in MyCourses.
 - Slides will be made available in MyCourses just before each lecture.
- Recorded Zoom-lectures will be available on MyCourses (unless the recording fails as it sometimes does).
- Optional textbook:
 - Gustafsson, F. (2018). Statistical Sensor Fusion. Studentlitteratur, 3rd edition.
 - Very complete (and more extensive) treatment of the subject
 - Not required to pass the course
 - Good reference for the future

Assessment and Grading

Assessment

To pass this course, you need to:

- pass the *written exam*,
- pass the *project*.
- get at least 3 points from *homeworks*.

Online Exam

- Online exam in MyCourses,
- Allowed aids: anything, but you cannot communicate with others, plagiarism will be checked.

Grading

- The grading scale is 0–5,
- The final grade is the maximum of the exam and project.

Exercise Sessions

- Exercise sessions are held on Tuesdays, 12:15 - 14.00 in Zoom, starting on Tuesday, Sep 15, 2020.
- In the exercise sessions, the teacher shows you hands on how to solve the exercises.
- You also have the chance to solve the exercises yourself.
- Pen & paper and computer exercises
 - Matlab or Octave recommended, Python is possible
- Exercise sessions are not mandatory but highly recommended, the exam questions are likely to be related to the exercises

Homeworks

- In the end of each exercise paper there is a homework.
 - You must complete at least 3 homeworks (out of 11) to pass the course.
 - From each homework exceeding 3 you can get 1/2 point to exam:

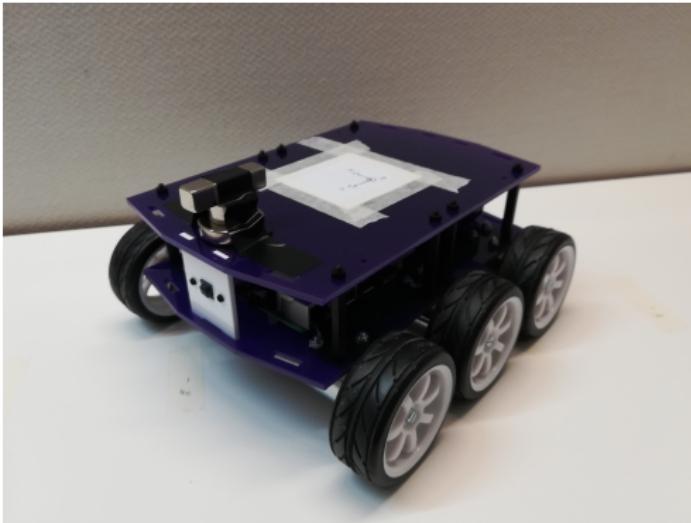
$$\text{extra exam points} = \max(0, (n - 3)/2),$$

where n is the number of returned homeworks.

- The homeworks need to be returned on MyCourses before the next exercise session day at 12:00.

Project work

- Track an **autonomous robot** using multiple sensors
- **Details** of the project work will be provided later.



Presemo Questionnaire

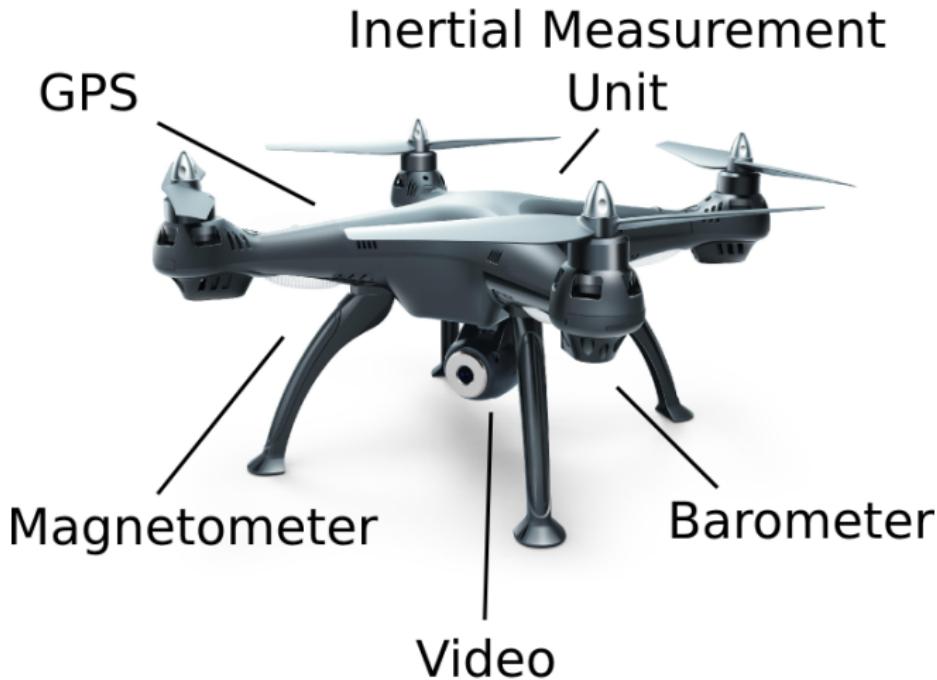
- We are using presemo on this course.
- Please use your computer or mobile phone and go to:

<http://presemo.aalto.fi/fusion>

Definition of Sensor Fusion

- One possible definition of **sensor fusion**:
"computational methodology which aims at combining the measurements from multiple sensors such that they jointly give more information on the measured system than any of the sensors alone."
- The important aspects are:
 - It is **computational methodology**.
 - Uses measurements from **multiple sensors**.
 - Attempts to use the **information** from all the sensors **jointly**.

Sensor Fusion Applications: Drones



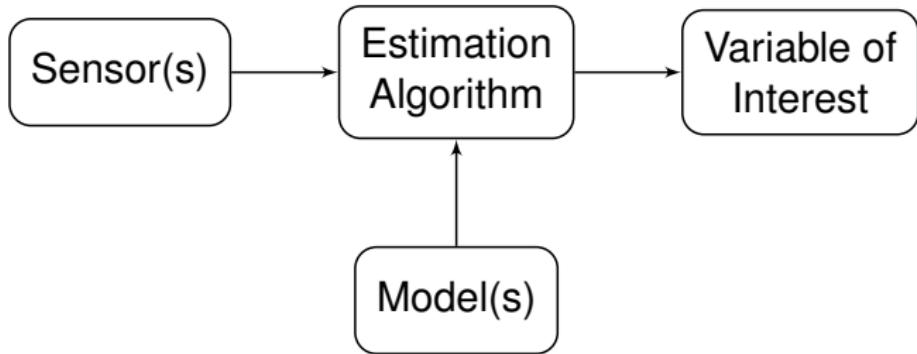
Sensor Fusion Applications: Autonomous Cars



Sensor Fusion Applications: Smartphones



The Components of Sensor Fusion



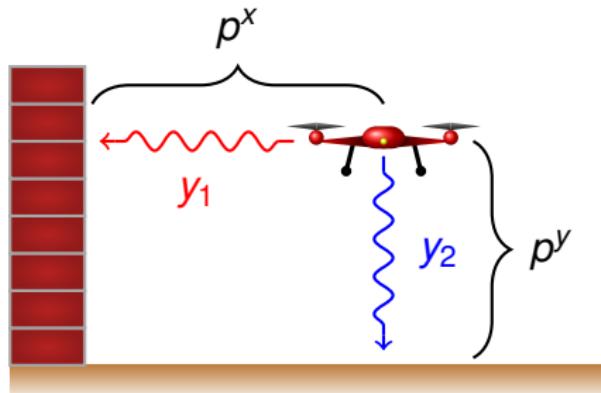
Model of a Drone (1)

- We measure y_1 with e.g. radar or ultrasound.
- We measure y_2 with e.g. radar or barometer.
- We wish to "fuse" the sensor measurements to get the location (p^x, p^y) .
- The model in this case is

$$y_1 = p^x + r_1,$$

$$y_2 = p^y + r_2. \quad (r_1 \text{ and } r_2 \text{ here denote measurement noises})$$

- Sensor fusion amounts to just $p^x \approx y_1$ and $p^y \approx y_2$.



Model of a Drone (2)

- We could also measure the distance y_3 to an additional tilted wall.
- The model now becomes

$$y_1 = p^x + r_1,$$

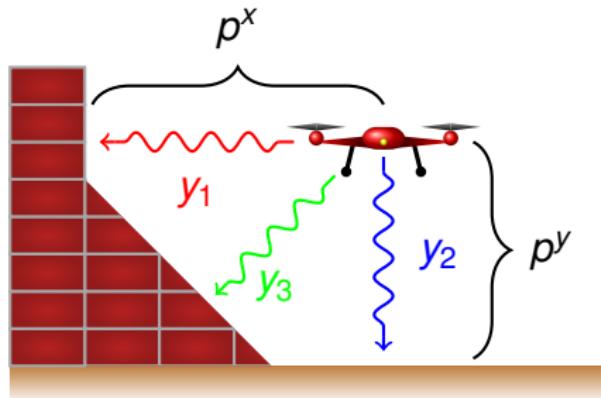
$$y_2 = p^y + r_2,$$

$$y_3 = \frac{1}{\sqrt{2}} (p^x - x_0) + \frac{1}{\sqrt{2}} p^y + r_3.$$

- In vector form:

$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{b} + \mathbf{r}.$$

- Linear least squares method gives $\mathbf{x} = (p^x, p^y)$.



Model of an Autonomous Car (1)

- We measure relative positions of M landmarks.
- We get $2M$ measurements ($M = 4$ here):

$$y_1 = s_1^x - p^x + r_1,$$

$$y_2 = s_1^y - p^y + r_2,$$

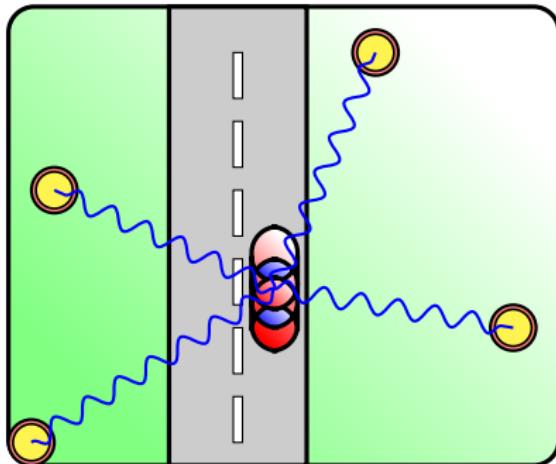
$$\vdots$$

$$y_{2M-1} = s_M^x - p^x + r_{2M-1},$$

$$y_{2M} = s_M^y - p^y + r_{2M}.$$

- Again leads to form

$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{b} + \mathbf{r}.$$



Model of an Autonomous Car (2)

- We only measure the range to each landmark.
- In that case we have

$$y_1^R = \sqrt{(s_1^x - p^x)^2 + (s_1^y - p^y)^2} + r_1^R,$$

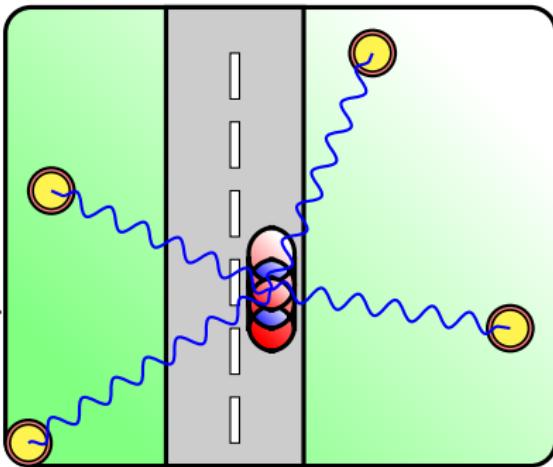
⋮

$$y_M^R = \sqrt{(s_M^x - p^x)^2 + (s_M^y - p^y)^2} + r_M^R.$$

- This is a non-linear model

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r}$$

- Non-linear least squares method is needed.



Dynamic Models

- The object of interest might also be moving.
- We can model time-continuity with a dynamic model.
- For example, we might have

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \mathbf{q}_n \quad (\text{here } \mathbf{q}_n \text{ is a noise process})$$

- More generally we get state-space models of the form

$$\begin{aligned}\mathbf{x}_n &= \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n, \\ \mathbf{y}_n &= \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n.\end{aligned}$$

- Can be coped with Kalman filters, extended/unscented Kalman filters, and particle filters.

Technical Contents of the Course

- Formulation of sensor fusion as a least squares problem.
- Solution methods for linear least squares problems.
- Solution methods for non-linear least squares problems.
- Solution methods for dynamic least squares (state-estimation) problems.
- Implementation of the methodology to robot platform.

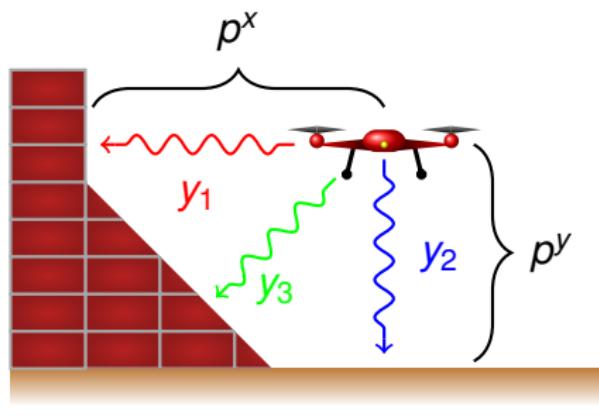


Summary (1)

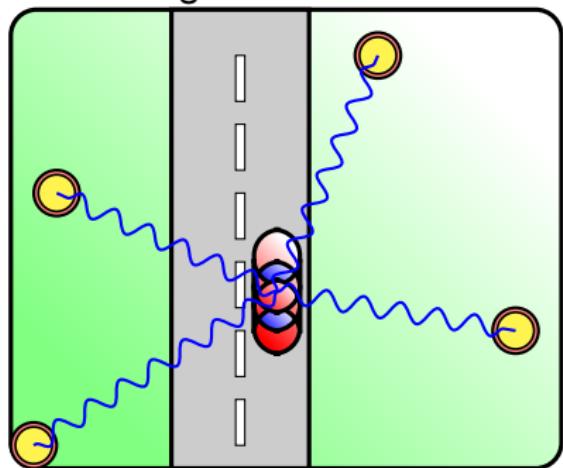
- Zoom lectures are on Fridays in 12:15-14:00
- Zoom exercises on Tuesdays in 12:15-14:00
- Teaching materials are lecture notes and slides on MyCourses.
- Project work starts later and it is about sensor fusion in a mobile robot.
- Exam is in early December and project work deadline just before Christmas.
- Homeworks earn points to exam.
- Sensor fusion is methodology for intelligent processing of measurements from multiple sensors.
- In practice, linear/non-linear least squares methods and Kalman/particle filtering methods.

Summary (2)

Typical models that we saw are the following:



$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{b} + \mathbf{r}$$



$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r}$$

Presemo Questionnaire

<http://presemo.aalto.fi/fusion>



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Sensors, Models, and Least Squares Criterion

Simo Särkkä

Aalto University

September 18, 2020

Contents

- 1 Intended Learning Outcomes and Recap
- 2 Components of Sensor Fusion
- 3 Mathematical Formulation
- 4 Summary

Intended Learning Outcomes

After this lecture, you will be able to:

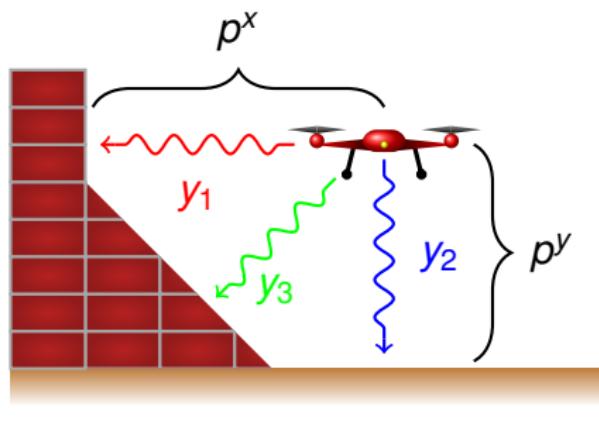
- Recognize and name the components of sensor fusion systems: sensors, models, estimation algorithms;
- Understand the notation used in sensor fusion (on this course);
- Describe and identify the purpose of an optimality criterion and cost functions;
- Understand what are least squares, weighted least squares, and regularized least squares criteria.

Recap (1)

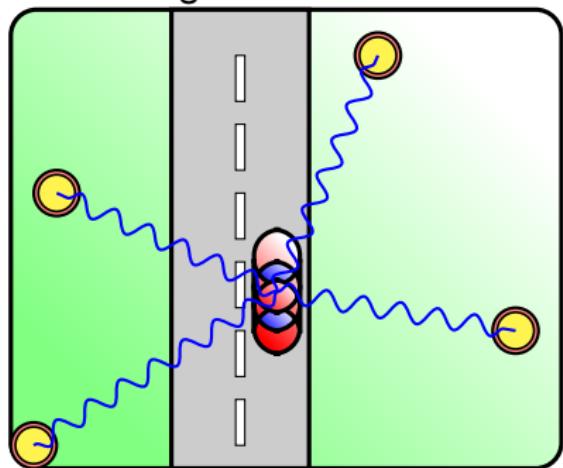
- Zoom lectures are on Fridays in 12:15-14:00
- Zoom exercises on Tuesdays in 12:15-14:00
- Teaching materials are lecture notes and slides on MyCourses.
- Project work starts later and it is about sensor fusion in a mobile robot.
- Exam is in early December and project work deadline just before Christmas.
- Homeworks earn points to exam.
- Sensor fusion is methodology for intelligent processing of measurements from multiple sensors.
- In practice, linear/non-linear least squares methods and Kalman/particle filtering methods.

Recap (2)

Typical models that we saw are the following:

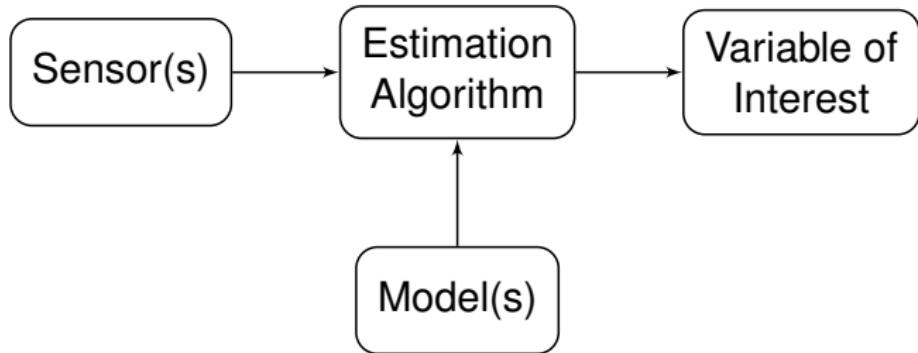


$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{b} + \mathbf{r}$$



$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r}$$

The Components of Sensor Fusion



Variable of Interest

Definition

One or more unknown static **quantities of interest**, parameters or a time-varying **state** of a dynamic system of interest that can be measured directly or indirectly.

Notation

- A single (scalar) static parameter is denoted x ,
- a vector of K static parameters is denoted as
$$\mathbf{x} = [x_1 \quad x_2 \quad \dots \quad x_K]^T,$$
- a scalar time-varying state is denoted $x_n = x(t_n)$,
- a vector time-varying state is denoted $\mathbf{x}_n = \mathbf{x}(t_n)$.

Sensors (1/3)

Definition

- Sensor is a device that provides a measurement related to the quantity of interest.
 - Usually, implemented as a device which converts a physical phenomenon into an electrical signal (Wilson, 2005) which is then further transformed into digital form.
-
- May measure the variable directly or indirectly
 - Measurement range and environmental conditions
 - Is affected by noise, biases, and uncertainty
 - May give measurements frequently or infrequently
 - Scalar or vector measurements

Sensors (2/3)

Notation

- A scalar measurement is denoted y_n
- A vector measurement is denoted \mathbf{y}_n
- n is a measurement number, sensor id, time, etc.

Sensors (3/3)

Examples

Sensor	Measurement	Application Examples
Accelerometer	Gravity, acceleration	Inertial navigation, activity tracking, screen rotation
Gyroscope	Rotational velocity	Inertial navigation, activity tracking
Magnetometer	Magnetic field strength	Inertial navigation, digital compass, object tracking
Radar	Range, bearing, speed	Target tracking, autonomous vehicles
LIDAR	Range, bearing, speed	Target tracking, autonomous vehicles, robotics
Ultrasound	Range	Robotics
Camera	Visual scene	Security systems, autonomous vehicles, robotics
Barometer	Air pressure	Inertial navigation, autonomous vehicles, robotics
GNSS	Position	Autonomous vehicles, aerospace applications
Strain gauge	Strain	Condition monitoring, scales

Definition

Describes how the **variable of interest** is observed by the **sensor** in a systematic way.

- May be very simple or very complex
- Takes noise, uncertainty, and other error sources into account
- Formulated using mathematics

Estimation Algorithm

Definition

Combines the **measurements from multiple sensors** by using the corresponding models to **estimate the quantities of interest** in some optimal sense.

- Combining multiple measurements increases the precision (on average)
- Measurements from different sensors can be incorporated
- Can account for the uncertainty of different measurements
- In this course the algorithms minimize a least squares cost criterion to achieve these.

A Basic Model

- The general form is:

Measurement = Function of Parameter(s) + Noise

- Mathematically:

$$y_n = g_n(\mathbf{x}) + r_n$$

- Anatomy:

- Measurement y_n is on the left hand side, and
- Function $g_n(\mathbf{x})$ of \mathbf{x} and a *noise* term r_n on the right side.

- This is called *sensor model*, *measurement model*, or *observation model*.

Measurement Noise

- Encodes thermal sensor noise, uncertainty, etc.
- r_n is modeled as a **random variable**, follows a probability density function (pdf)

$$r_n \sim p(r_n)$$

- For now, we assume zero-mean, independent random variables with variance $\sigma_{r,n}^2$

$$\mathbb{E}\{r_n\} = 0,$$

$$\text{var}\{r_n\} = \mathbb{E}\{r_n^2\} - (\mathbb{E}\{r_n\})^2 = \sigma_{r,n}^2,$$

$$\text{Cov}\{r_m, r_n\} = \mathbb{E}\{r_m r_n\} - \mathbb{E}\{r_m\} \mathbb{E}\{r_n\} = 0 \quad (m \neq n)$$

Vector Model

- Extending the basic model for *vector-valued* measurements:

$$\mathbf{y}_n = \mathbf{g}_n(\mathbf{x}) + \mathbf{r}_n,$$

- \mathbf{y}_n and \mathbf{r}_n are d_y -dimensional column vectors
- \mathbf{r}_n is a multivariate random variable with pdf

$$\mathbf{r}_n \sim p(\mathbf{r}_n)$$

- Assume zero-mean, independent random variables with covariance \mathbf{R}_n

$$E\{\mathbf{r}_n\} = 0,$$

$$\text{Cov}\{\mathbf{r}_n\} = E\{\mathbf{r}_n \mathbf{r}_n^T\} - E\{\mathbf{r}_n\} E\{\mathbf{r}_n\}^T = \mathbf{R}_n,$$

$$\text{Cov}\{\mathbf{r}_m, \mathbf{r}_n\} = E\{\mathbf{r}_m \mathbf{r}_n^T\} - E\{\mathbf{r}_m\} E\{\mathbf{r}_n\}^T = 0 \quad (m \neq n)$$

Multiple Measurements

- Sensor fusion requires multiple sensors, repeated measurements, or both
- In the terminology of the measurement model, they can be regarded the same: y_1, y_2, \dots, y_N or $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N$
- We denote a **set of measurements**:
 - $y_{1:N} = \{y_1, y_2, \dots, y_N\}$ for the scalar case
 - $\mathbf{y}_{1:N} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N\}$ for the vector case
- Examples: Sensor networks, sensor arrays, multi-view imaging, etc.

Measurement Stacking: Scalar Case

- Remember: $y_n = g_n(\mathbf{x}) + r_n$
- Given the measurements $y_{1:N}$, we can write:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_N(\mathbf{x}) \end{bmatrix} + \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix}$$

- Compact notation for all measurements:

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r}.$$

- Covariance for \mathbf{r} : $\text{Cov}\{\mathbf{r}\} = \mathbf{R} = \begin{bmatrix} \sigma_{r,1}^2 & 0 & \dots & 0 \\ 0 & \sigma_{r,2}^2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \sigma_{r,N}^2 \end{bmatrix}$

Measurement Stacking: Vector Case

- Vector case: $\mathbf{y}_n = \mathbf{g}_n(\mathbf{x}) + \mathbf{r}_n$

- Stacked notation:

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} = \begin{bmatrix} \mathbf{g}_1(\mathbf{x}) \\ \mathbf{g}_2(\mathbf{x}) \\ \vdots \\ \mathbf{g}_N(\mathbf{x}) \end{bmatrix} + \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_N \end{bmatrix}$$

- Hence,

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r}.$$

where $\text{Cov}\{\mathbf{r}\} = \mathbf{R} = \begin{bmatrix} \mathbf{R}_1 & 0 & \dots & 0 \\ 0 & \mathbf{R}_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \mathbf{R}_N \end{bmatrix}$.

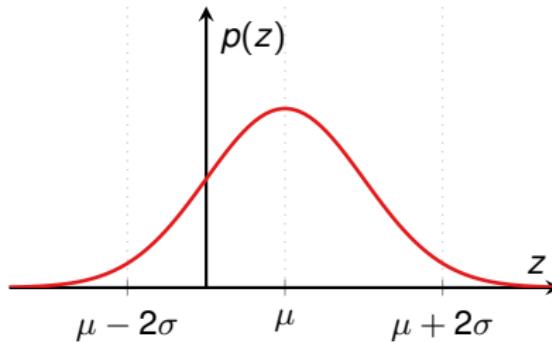
Gaussian Measurement Noise

- Noise is often assumed to be Gaussian, i.e.

$$p(\mathbf{r}) = \frac{1}{(2\pi)^{M/2} |\mathbf{R}|^{1/2}} e^{-\frac{1}{2}\mathbf{r}^T \mathbf{R}^{-1} \mathbf{r}}$$

- Compact notation $p(\mathbf{r}) = \mathcal{N}(\mathbf{r}; \mathbf{0}, \mathbf{R})$, where

$$\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{M/2} |\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{z}-\boldsymbol{\mu})}$$



Cost Functions (1/2)

- An optimality criterion is required to develop an estimation algorithm
- We focus on algorithms that minimize a cost function of the error

$$\hat{\mathbf{x}} = \operatorname{argmin}_{\mathbf{x}} J(\mathbf{x})$$

where

- $\hat{\mathbf{x}}$ denotes the estimate of \mathbf{x}
- $J(\mathbf{x})$ is the cost function
- $\operatorname{argmin}_{\mathbf{x}} J(\mathbf{x})$ denotes “the argument \mathbf{x} that minimizes $J(\mathbf{x})$ ”
- The error is given by the difference between the measurement and the output predicted by \mathbf{x}

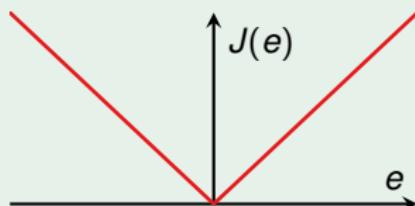
$$e_n = y_n - g_n(\mathbf{x})$$

Cost Functions (2/2)

Absolute Error

Penalizes all errors equally:

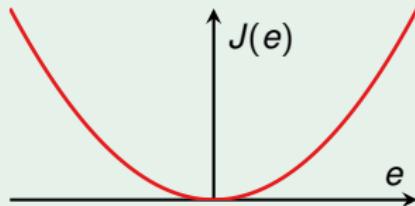
$$|e_n| = |y_n - g_n(\mathbf{x})|,$$



Quadratic Error

Penalizes large errors more than small ones:

$$e_n^2 = (y_n - g_n(\mathbf{x}))^2.$$



Least Squares (1/2)

- The quadratic cost is much more common
- Closely related to Gaussian measurement noise
- Minimizing the quadratic cost function is the least squares method
- Cost function for N scalar measurements

$$y_{1:N} = \{y_1, y_2, \dots, y_N\}$$

$$J_{\text{LS}}(\mathbf{x}) = \sum_{n=1}^N e_n^2 = \sum_{n=1}^N (y_n - g_n(\mathbf{x}))^2$$

Least Squares (2/2)

- Quadratic error for vector measurements

$$e_n^2 = (\mathbf{y}_n - \mathbf{g}_n(\mathbf{x}))^\top (\mathbf{y}_n - \mathbf{g}_n(\mathbf{x}))$$

- Cost function for N vector measurements

$$\mathbf{y}_{1:N} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N\}$$

$$J_{\text{LS}}(\mathbf{x}) = \sum_{n=1}^N (\mathbf{y}_n - \mathbf{g}_n(\mathbf{x}))^\top (\mathbf{y}_n - \mathbf{g}_n(\mathbf{x}))$$

- Quadratic error and cost function for stacked (batch) notation

$$J_{\text{LS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top (\mathbf{y} - \mathbf{g}(\mathbf{x}))$$

Weighted Least Squares (1/2)

- How to include confidence in sensor readings?
- Weighted least squares (WLS) cost function:

$$J_{\text{WLS}}(\mathbf{x}) = \sum_{n=1}^N w_n (y_n - g_n(\mathbf{x}))^2,$$

where $w_n > 0$ is a weighing factor for the n th measurement

- WLS vector cost function:

$$J_{\text{WLS}}(\mathbf{x}) = \sum_{n=1}^N (\mathbf{y}_n - \mathbf{g}_n(\mathbf{x}))^\top \mathbf{W}_n (\mathbf{y}_n - \mathbf{g}_n(\mathbf{x})),$$

where \mathbf{W}_n is a positive-definite weighing matrix

Weighted Least Squares (2/2)

- WLS stacked cost function:

$$J_{\text{WLS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{W} (\mathbf{y} - \mathbf{g}(\mathbf{x}))$$

where \mathbf{W} is the positive-definite weighing matrix

$$\mathbf{W} = \begin{bmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & w_N \end{bmatrix} \quad \text{or} \quad \mathbf{W} = \begin{bmatrix} \mathbf{W}_1 & 0 & \dots & 0 \\ 0 & \mathbf{W}_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \mathbf{W}_N \end{bmatrix}$$

- Choice of w_n or \mathbf{W}_n is in principle arbitrary
- In practice, good choices are

$$w_n = 1/\sigma_{r,n}^2 \text{ and } \mathbf{W}_n = \mathbf{R}_n^{-1}$$

Regularized Least Squares

- In plain (weighted) least squares we find an estimate that best explains the measurements.
- In regularized least squares we add a penalty term in the estimate.
- The penalty term can force the estimate to be "small" or close to certain "a priori" known value.
- The general form of regularized least squares (ReLS) that we use is

$$J_{\text{ReLS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})) + (\mathbf{x} - \mathbf{m})^\top \mathbf{P}^{-1} (\mathbf{x} - \mathbf{m}).$$

- Regularized least squares can also be used to formulate dynamic estimation methods.

Summary

- Sensor fusion involves three components:
 - ① **Sensor:** Measures a variable of interest, directly or indirectly
 - ② **Model:** A mathematical formulation that relates the variables of interest to the measurements
 - ③ **Estimation Algorithm:** Combines the measurements and models to estimate the variables of interest
- Multiple measurements and multidimensional measurements can be written in **the same vector notation.**
- **The least squares method** is a good way for deriving estimators.
- **(Plain) least squares, weighted least squares, and regularized least squares** are useful criteria for estimators.



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Static Linear Models and Linear Least Squares

Simo Särkkä

Aalto University

September 25, 2020

Contents

- 1 Intended Learning Outcomes and Recap
- 2 Linear Scalar and Vector Models
- 3 Least Squares for General Linear Model
- 4 Weighted Least Squares
- 5 Regularized Least Squares
- 6 Sequential Least Squares
- 7 Summary

Intended Learning Outcomes

After this lecture, you will be able to:

- Identify and construct scalar and vector linear models;
- apply and derive (weighted, regularized, and sequential) linear least squares estimators;
- investigate the properties of linear least squares estimators.

Recap

- Sensor fusion involves three components:
 - ① **Sensor:** Measures a variable of interest, directly or indirectly
 - ② **Model:** A mathematical formulation that relates the variables of interest to the measurements
 - ③ **Estimation Algorithm:** Combines the measurements and models to estimate the variables of interest
- Multiple measurements and multidimensional measurements can be written in **the same vector notation.**
- **The least squares method** is a good way for deriving estimators.
- **(Plain) least squares, weighted least squares, and regularized least squares** are useful criteria for estimators.

Scalar Model: Model & Cost Function

- Many sensors measure (a scaled) version of a **single unknown x**

$$y_n = gx + r_n,$$

with $E\{r_n\} = 0$ and $\text{var}\{r_n\} = \sigma_{r,n}^2$

- The **error** for one measurement is

$$e_n = y_n - gx$$

and the **least squares cost function** is given by

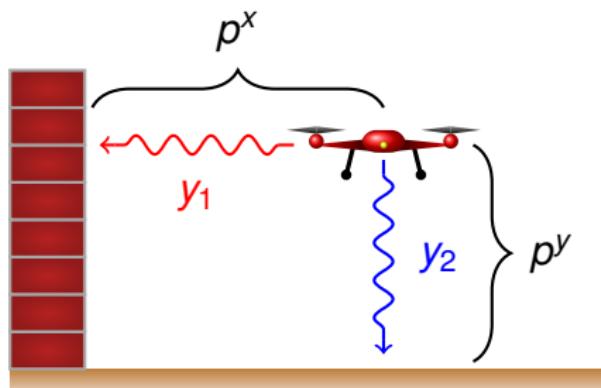
$$J_{\text{LS}}(x) = \sum_{n=1}^N (y_n - gx)^2$$

Scalar Model: Example

Example

- Radar measures the time difference between the sent and reflected signals.
- Double distance divided by the speed of light
 $\tau = 2p^x/c$, where
 $c = 299792458 \text{ m/s}$.
- The measurement model is (beware of the notation!)

$$y_n = \frac{2}{c} p^x + r_n, \quad n = 1, \dots, N.$$



Scalar Model: Minimizing the Cost

- The derivative is given by

$$\frac{\partial J_{\text{LS}}(x)}{\partial x} = -2g \sum_{n=1}^N y_n + 2Ng^2 x$$

- Setting the derivative to zero and solving for x yields

$$\hat{x}_{\text{LS}} = \frac{1}{Ng} \sum_{n=1}^N y_n.$$

- This is the least squares estimator for the model

$$y_n = gx + r_n.$$

Scalar Model: Estimator Properties

- What are the estimator's statistical properties?
- Expected value:

$$E\{\hat{x}_{LS}\} = x + \sum_{n=1}^N E\{r_n\} = x$$

- Variance:

$$\text{var}\{\hat{x}\} = \frac{1}{N^2 g^2} \sum_{n=1}^N \sigma_{r,n}^2$$

and when $\sigma_{r,n}^2 = \sigma^2$ we get

$$\text{var}\{\hat{x}\} = \frac{\sigma^2}{N g^2} \quad \text{and} \quad \text{std}\{\hat{x}\} = \frac{1}{\sqrt{N}} \frac{\sigma}{|g|}.$$

- The expectation of \hat{x} is $x \Rightarrow$ estimator is unbiased.

Scalar Model: Example

Example

Let us consider the wall-distance measurement model and assume that we estimate p^x with N measurements:

$$\hat{x}_1 = \frac{c}{2N} \sum_{n=1}^N y_n.$$

This estimator is unbiased. Further assume that the standard deviation of the measurement is $\sigma = 10^{-9}$ s (1 nanosecond). Then the standard deviation of the estimator is

$$\text{std}\{\hat{x}_1\} = \frac{1}{\sqrt{N}} \frac{c\sigma}{2}.$$

With a single measurement we get the error of 15 cm whereas by averaging 100 measurements the error drops to 1.5 cm.

Vector Models

- Scalar observations, several parameters x_1, x_2, \dots, x_K :

$$\begin{aligned}y_n &= g_1 x_1 + g_2 x_2 + \cdots + g_K x_K + r_n \\&= \mathbf{g}\mathbf{x} + r_n\end{aligned}$$

- Slightly more generally:

$$y_n = \mathbf{g}_n \mathbf{x} + r_n$$

- Stacking measurements together gives:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_N \end{bmatrix} \mathbf{x} + \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix}$$

- This has the general form

$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{r}, \text{ with } \text{Cov}\{\mathbf{r}\} = \mathbf{R} = \text{diag}(\sigma_{r,1}^2, \dots, \sigma_{r,N}^2).$$

Vector Models (cont.)

- Vector observations, several parameters:

$$\mathbf{y}_n = \begin{bmatrix} g_{11} & g_{12} & \dots & g_{1K} \\ g_{21} & g_{22} & \dots & g_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ g_{d_y 1} & g_{d_y 2} & \dots & g_{d_y K} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix} + \mathbf{r}_n$$
$$= \mathbf{G}_n \mathbf{x} + \mathbf{r}_n,$$

- Batch notation:

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} = \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{G}_2 \\ \vdots \\ \mathbf{G}_N \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_N \end{bmatrix}$$

- In compact notation we again get the same general form:

$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{r}, \text{ with } \text{Cov}\{\mathbf{r}\} = \mathbf{R} = \text{diag}(\mathbf{R}_1, \dots, \mathbf{R}_N).$$

General Linear Model: Definition

- General form of a linear models:

$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{r},$$

with $E\{\mathbf{r}\} = 0$ and $\text{Cov}\{\mathbf{r}\} = \mathbf{R}$.

- This is the general linear model, both the scalar and vector cases can be expressed in this way.

Affine Models

- We might also have a constant bias \mathbf{b} in the model:

$$\mathbf{y} = \mathbf{Gx} + \mathbf{b} + \mathbf{r}.$$

- We can now compute a modified measurement $\tilde{\mathbf{y}} = \mathbf{y} - \mathbf{b}$ and rewrite this as

$$\tilde{\mathbf{y}} = \mathbf{Gx} + \mathbf{r}.$$

- Thus is again a general linear model.

Example: Localizing a Drone

- Recall the drone model:

$$y_1 = p^x + r_1,$$

$$y_2 = p^y + r_2,$$

$$y_3 = \frac{1}{\sqrt{2}} (p^x - x_0) + \frac{1}{\sqrt{2}} p^y + r_3.$$

- It has the affine form:

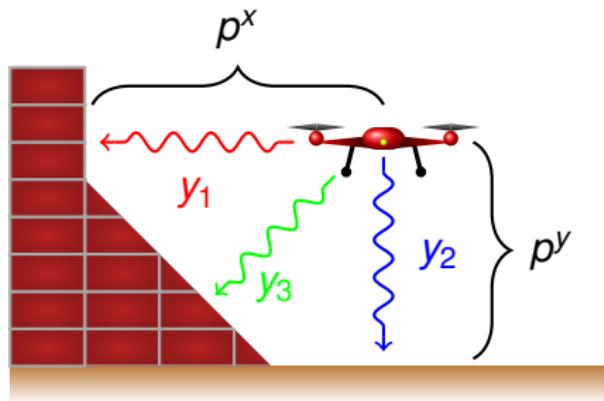
$$\mathbf{y} = \mathbf{G} \mathbf{x} + \mathbf{b} + \mathbf{r}.$$

- Can be reduced to linear model by defining

$$\tilde{y}_1 = y_1,$$

$$\tilde{y}_2 = y_2,$$

$$\tilde{y}_3 = y_3 + \frac{1}{\sqrt{2}} x_0.$$



Example: Localizing a Car

- We have:

$$y_1 = s_1^x - p^x + r_1,$$

$$y_2 = s_1^y - p^y + r_2,$$

$$\vdots$$

$$y_{2M} = s_M^y - p^y + r_{2M}.$$

- Again leads to form

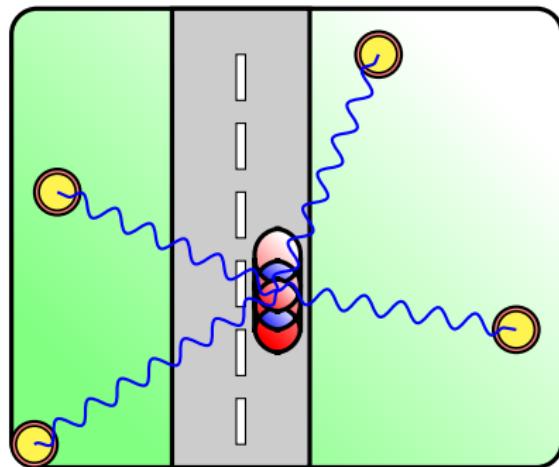
$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{b} + \mathbf{r}.$$

- We can now define

$$\tilde{y}_1 = y_1 - s_1^x,$$

$$\vdots$$

$$\tilde{y}_{2M} = y_{2M} - s_M^y.$$



General Linear Model: Least Squares (1)

- The least squares cost function to minimize:

$$\begin{aligned} J_{\text{LS}}(\mathbf{x}) &= (\mathbf{y} - \mathbf{Gx})^T(\mathbf{y} - \mathbf{Gx}) \\ &= \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{Gx} - \mathbf{x}^T \mathbf{G}^T \mathbf{y} + \mathbf{x}^T \mathbf{G}^T \mathbf{Gx} \end{aligned}$$

- Some vector calculus identities (when \mathbf{A} is symmetric):

$$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

$$\frac{\partial \mathbf{x}^T \mathbf{Ax}}{\partial \mathbf{x}} = 2\mathbf{Ax}$$

General Linear Model: Least Squares (2)

- The least squares estimator for the general linear model is

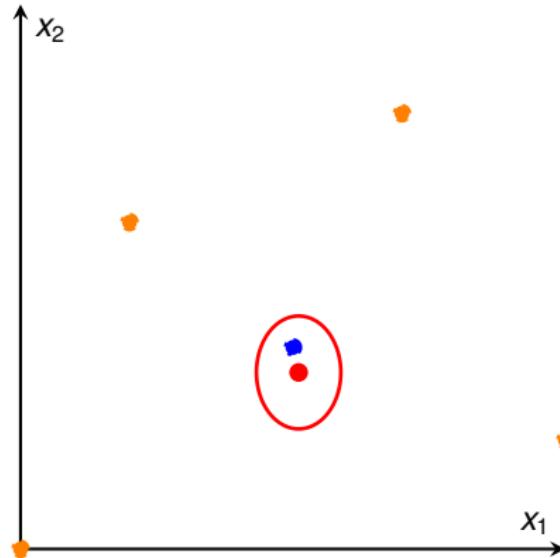
$$\hat{\mathbf{x}}_{\text{LS}} = (\mathbf{G}^T \mathbf{G})^{-1} \mathbf{G}^T \mathbf{y}$$

- Its statistical properties are

$$E\{\hat{\mathbf{x}}_{\text{LS}}\} = \mathbf{x}$$

$$\text{Cov}\{\hat{\mathbf{x}}_{\text{LS}}\} = (\mathbf{G}^T \mathbf{G})^{-1} \mathbf{G}^T \mathbf{R} \mathbf{G} ((\mathbf{G}^T \mathbf{G})^{-1})^T.$$

Example: Localizing a Car (1)



Weighted Linear Least Squares (1)

- Recall the general linear model:

$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{r}.$$

with $E\{\mathbf{r}\} = 0$ and $\text{Cov}\{\mathbf{r}\} = \mathbf{R}$.

- Weighted least squares cost function:

$$J_{WLS}(\mathbf{x}) = (\mathbf{y} - \mathbf{G}\mathbf{x})^T \mathbf{R}^{-1} (\mathbf{y} - \mathbf{G}\mathbf{x})$$

- We can now derive the estimator in the same way as for (plain) least squares.

Weighted Linear Least Squares (2)

- Weighted linear least squares estimator:

$$\hat{\mathbf{x}}_{WLS} = (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G})^{-1} \mathbf{G}^T \mathbf{R}^{-1} \mathbf{y}.$$

- Properties:

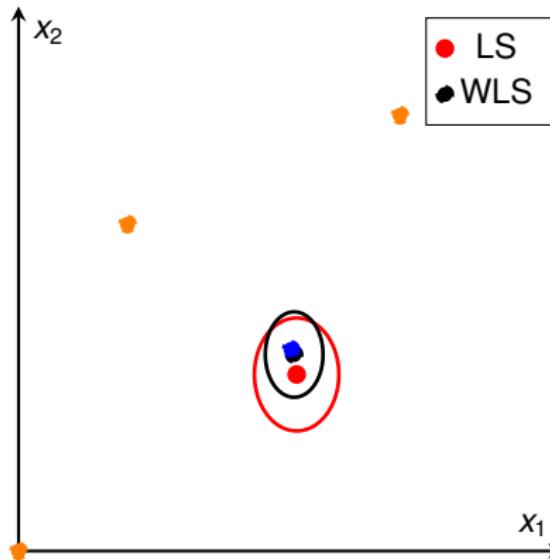
$$E\{\hat{\mathbf{x}}_{WLS}\} = \mathbf{x}$$

$$\text{Cov}\{\hat{\mathbf{x}}_{WLS}\} = (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G})^{-1}$$

- It can be shown that $\mathbf{W} = \mathbf{R}^{-1}$ minimizes $\text{Cov}\{\hat{\mathbf{x}}_{WLS}\}$ over all choices for \mathbf{W} and in this case

$$\text{Cov}\{\hat{\mathbf{x}}_{WLS}\} \leq \text{Cov}\{\hat{\mathbf{x}}_{LS}\}$$

Example: Localizing a Car (2)



Regularized Linear Least Squares (1/2)

- The regularized least squares criterion

$$J_{\text{ReLS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{G}\mathbf{x})^T \mathbf{R}^{-1} (\mathbf{y} - \mathbf{G}\mathbf{x}) + (\mathbf{x} - \mathbf{m})^T \mathbf{P}^{-1} (\mathbf{x} - \mathbf{m}).$$

- The regularized linear least squares estimator

$$\hat{\mathbf{x}}_{\text{ReLS}} = (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G} + \mathbf{P}^{-1})^{-1} (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{y} + \mathbf{P}^{-1} \mathbf{m}).$$

- The expectation is not \mathbf{x} !
- The covariance of the estimator is

$$\text{Cov}\{\hat{\mathbf{x}}_{\text{ReLS}}\} = (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G} + \mathbf{P}^{-1})^{-1}.$$

- The covariance is always smaller (or equal) to the WLS estimator.

Regularized Linear Least Squares (2/2)

- By using the matrix inversion formula we can write

$$(\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G} + \mathbf{P}^{-1})^{-1} = \mathbf{P} - \mathbf{P} \mathbf{G}^T (\mathbf{G} \mathbf{P} \mathbf{G}^T + \mathbf{R})^{-1} \mathbf{G} \mathbf{P}.$$

- This gives

$$\mathbf{K} = \mathbf{P} \mathbf{G}^T (\mathbf{G} \mathbf{P} \mathbf{G}^T + \mathbf{R})^{-1},$$

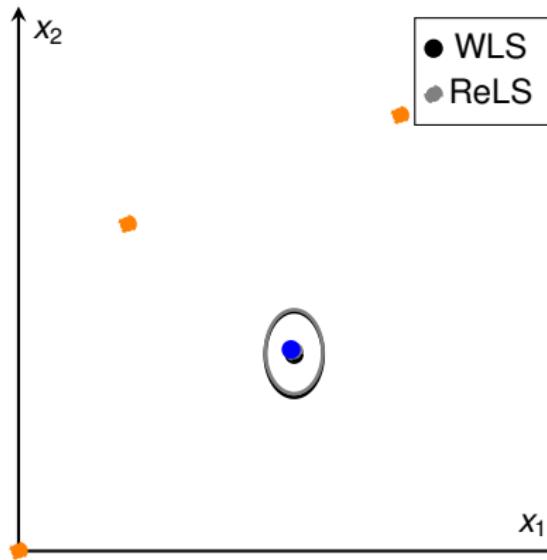
$$\hat{\mathbf{x}}_{\text{ReLS}} = \mathbf{m} + \mathbf{K}(\mathbf{y} - \mathbf{G}\mathbf{m}),$$

$$\text{Cov}\{\hat{\mathbf{x}}_{\text{ReLS}}\} = \mathbf{P} - \mathbf{K}(\mathbf{G} \mathbf{P} \mathbf{G}^T + \mathbf{R}) \mathbf{K}^T.$$

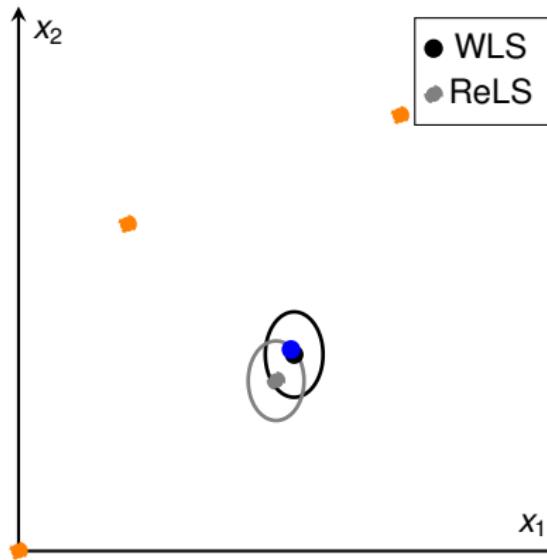
- Finally, we can always rewrite regularized least squares as weighted least squares:

$$\begin{aligned} J_{\text{ReLS}}(\mathbf{x}) &= (\mathbf{y} - \mathbf{G}\mathbf{x})^T \mathbf{R}^{-1} (\mathbf{y} - \mathbf{G}\mathbf{x}) + (\mathbf{m} - \mathbf{x})^T \mathbf{P}^{-1} (\mathbf{m} - \mathbf{x}) \\ &= \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{G} \\ \mathbf{I} \end{bmatrix} \mathbf{x} \right)^T \begin{bmatrix} \mathbf{R}^{-1} & 0 \\ 0 & \mathbf{P}^{-1} \end{bmatrix} \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{G} \\ \mathbf{I} \end{bmatrix} \mathbf{x} \right). \end{aligned}$$

Example: Localizing a Car (3)



Example: Localizing a Car (4)



Sequential Linear Least Squares (1/2)

- In many cases, the sensor data arrives sequentially at the estimator.
- Assume that we have calculated the weighted least squares (WLS) estimate using $\mathbf{y}_{1:n-1} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{n-1}\}$:

$$\hat{\mathbf{x}}_{n-1} = (\mathbf{G}_{1:n-1}^T \mathbf{R}_{1:n-1}^{-1} \mathbf{G}_{1:n-1})^{-1} \mathbf{G}_{1:n-1}^T \mathbf{R}_{1:n-1}^{-1} \mathbf{y}_{1:n-1},$$

$$\text{Cov}\{\hat{\mathbf{x}}_{k-1}\} = (\mathbf{G}_{1:n-1}^T \mathbf{R}_{1:n-1}^{-1} \mathbf{G}_{1:n-1})^{-1} = \mathbf{P}_{n-1}.$$

- We can now rewrite the WLS the cost function in sequential form

$$\begin{aligned} J_{\text{SLS}}(\mathbf{x}) &= (\mathbf{y}_{1:n-1} - \mathbf{G}_{1:n-1} \mathbf{x})^T \mathbf{R}_{1:n-1}^{-1} (\mathbf{y}_{1:n-1} - \mathbf{G}_{1:n-1} \mathbf{x}) \\ &\quad + (\mathbf{y}_n - \mathbf{G}_n \mathbf{x})^T \mathbf{R}_n^{-1} (\mathbf{y}_n - \mathbf{G}_n \mathbf{x}). \end{aligned}$$

Sequential Linear Least Squares (2/2)

- Setting gradient to zero and substituting the already computed result gives:

$$\begin{aligned}\hat{\mathbf{x}}_n &= (\mathbf{G}_{1:n-1}^T \mathbf{R}_{1:n-1}^{-1} \mathbf{G}_{1:n-1} + \mathbf{G}_n^T \mathbf{R}_n^{-1} \mathbf{G}_n)^{-1} \\ &\quad \times (\mathbf{G}_{1:n-1}^T \mathbf{R}_{1:n-1}^{-1} \mathbf{y}_{1:n-1} + \mathbf{G}_n^T \mathbf{R}_n^{-1} \mathbf{y}_n) \\ &= (\mathbf{P}_{n-1}^{-1} + \mathbf{G}_n^T \mathbf{R}_n^{-1} \mathbf{G}_n)^{-1} (\mathbf{G}_{1:n-1}^T \mathbf{R}_{1:n-1}^{-1} \mathbf{y}_{1:n-1} + \mathbf{G}_n^T \mathbf{R}_n^{-1} \mathbf{y}_n).\end{aligned}$$

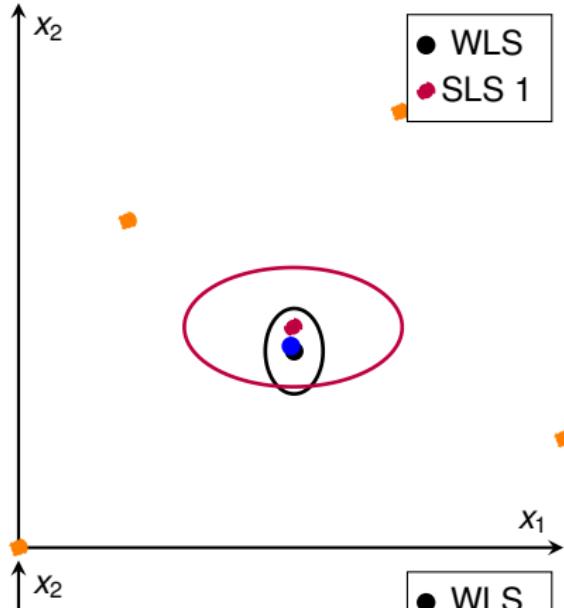
- Using matrix inversion formula gives

$$\begin{aligned}\mathbf{K}_n &= \mathbf{P}_{n-1} \mathbf{G}_n^T (\mathbf{G}_n \mathbf{P}_{n-1} \mathbf{G}_n^T + \mathbf{R}_n)^{-1}, \\ \hat{\mathbf{x}}_n &= \hat{\mathbf{x}}_{n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{G}_n \hat{\mathbf{x}}_{n-1}),\end{aligned}$$

$$\text{Cov}\{\hat{\mathbf{x}}_n\} = \mathbf{P}_{n-1} - \mathbf{K}_n (\mathbf{G}_n \mathbf{P}_{n-1} \mathbf{G}_n^T + \mathbf{R}_n) \mathbf{K}_n^T = \mathbf{P}_n.$$

- This is now a recursion for the estimates.
- Regularized least squares results from $\hat{\mathbf{x}}_0 = \mathbf{m}$ and $\mathbf{P}_0 = \mathbf{P}$.

Example: Localizing a Car (5)



Summary (1)

- The general linear model is given by

$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{r}, \quad E\{\mathbf{r}\} = 0, \quad \text{Cov}\{\mathbf{r}\} = \mathbf{R}$$

- Affine models can be tackled by rewriting

$$\begin{aligned}\mathbf{y} &= \mathbf{G}\mathbf{x} + \mathbf{b} + \mathbf{r}, \\ \underbrace{\mathbf{y} - \mathbf{b}}_{\tilde{\mathbf{y}}} &= \mathbf{G}\mathbf{x} + \mathbf{r}.\end{aligned}$$

- Different least squares estimators:

$$\hat{\mathbf{x}}_{LS} = (\mathbf{G}^T \mathbf{G})^{-1} \mathbf{G}^T \mathbf{y},$$

$$\hat{\mathbf{x}}_{WLS} = (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G})^{-1} \mathbf{G}^T \mathbf{R}^{-1} \mathbf{y},$$

$$\hat{\mathbf{x}}_{ReLS} = (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G} + \mathbf{P}^{-1})^{-1} (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{y} + \mathbf{P}^{-1} \mathbf{m}).$$

- We also computed their expectations and covariances.

Summary (2)

- Alternative form of regularized least squares estimator:

$$\mathbf{K} = \mathbf{P}\mathbf{G}^T(\mathbf{G}\mathbf{P}\mathbf{G}^T + \mathbf{R})^{-1},$$

$$\hat{\mathbf{x}}_{\text{ReLS}} = \mathbf{m} + \mathbf{K}(\mathbf{y} - \mathbf{G}\mathbf{m}),$$

$$\text{Cov}\{\hat{\mathbf{x}}_{\text{ReLS}}\} = \mathbf{P} - \mathbf{K}(\mathbf{G}\mathbf{P}\mathbf{G}^T + \mathbf{R})\mathbf{K}^T.$$

- Sequential (weighted/regularized) least squares estimator:

$$\mathbf{K}_n = \mathbf{P}_{n-1}\mathbf{G}_n^T(\mathbf{G}_n\mathbf{P}_{n-1}\mathbf{G}_n^T + \mathbf{R}_n)^{-1},$$

$$\hat{\mathbf{x}}_n = \hat{\mathbf{x}}_{n-1} + \mathbf{K}_n(\mathbf{y}_n - \mathbf{G}_n\hat{\mathbf{x}}_{n-1}),$$

$$\mathbf{P}_n = \mathbf{P}_{n-1} - \mathbf{K}_n(\mathbf{G}_n\mathbf{P}_{n-1}\mathbf{G}_n^T + \mathbf{R}_n)\mathbf{K}_n^T.$$



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Static Nonlinear Models, Gradient Descent, and Gauss–Newton

Simo Särkkä

Aalto University

October 2, 2020

Contents

- 1 Intended Learning Outcomes and Recap
- 2 Static Nonlinear Models
- 3 Gradient Descent Algorithm
- 4 Gauss–Newton Algorithm
- 5 Summary

Intended Learning Outcomes

After this lecture, you will be able to:

- Identify the need for non-linear models;
- understand the principles of gradient decent and Gauss–Newton methods;
- apply gradient decent and Gauss–Newton methods to nonlinear sensor fusion problems.

Recap (1)

- The general linear model is given by

$$\mathbf{y} = \mathbf{G}\mathbf{x} + \mathbf{r}, \quad E\{\mathbf{r}\} = 0, \quad \text{Cov}\{\mathbf{r}\} = \mathbf{R}$$

- Affine models can be tackled by rewriting

$$\begin{aligned}\mathbf{y} &= \mathbf{G}\mathbf{x} + \mathbf{b} + \mathbf{r}, \\ \underbrace{\mathbf{y} - \mathbf{b}}_{\tilde{\mathbf{y}}} &= \mathbf{G}\mathbf{x} + \mathbf{r}.\end{aligned}$$

- Different least squares estimators:

$$\hat{\mathbf{x}}_{LS} = (\mathbf{G}^T \mathbf{G})^{-1} \mathbf{G}^T \mathbf{y},$$

$$\hat{\mathbf{x}}_{WLS} = (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G})^{-1} \mathbf{G}^T \mathbf{R}^{-1} \mathbf{y},$$

$$\hat{\mathbf{x}}_{ReLS} = (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{G} + \mathbf{P}^{-1})^{-1} (\mathbf{G}^T \mathbf{R}^{-1} \mathbf{y} + \mathbf{P}^{-1} \mathbf{m}).$$

- We also computed their expectations and covariances.

Recap (2)

- Alternative form of regularized least squares estimator:

$$\mathbf{K} = \mathbf{P}\mathbf{G}^T(\mathbf{G}\mathbf{P}\mathbf{G}^T + \mathbf{R})^{-1},$$

$$\hat{\mathbf{x}}_{\text{ReLS}} = \mathbf{m} + \mathbf{K}(\mathbf{y} - \mathbf{G}\mathbf{m}),$$

$$\text{Cov}\{\hat{\mathbf{x}}_{\text{ReLS}}\} = \mathbf{P} - \mathbf{K}(\mathbf{G}\mathbf{P}\mathbf{G}^T + \mathbf{R})\mathbf{K}^T.$$

- Sequential (weighted/regularized) least squares estimator:

$$\mathbf{K}_n = \mathbf{P}_{n-1}\mathbf{G}_n^T(\mathbf{G}_n\mathbf{P}_{n-1}\mathbf{G}_n^T + \mathbf{R}_n)^{-1},$$

$$\hat{\mathbf{x}}_n = \hat{\mathbf{x}}_{n-1} + \mathbf{K}_n(\mathbf{y}_n - \mathbf{G}_n\hat{\mathbf{x}}_{n-1}),$$

$$\mathbf{P}_n = \mathbf{P}_{n-1} - \mathbf{K}_n(\mathbf{G}_n\mathbf{P}_{n-1}\mathbf{G}_n^T + \mathbf{R}_n)\mathbf{K}_n^T.$$

Static Nonlinear Models

- Linear models have closed form solutions, but are limited in many cases
- General nonlinear model has the form:

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r},$$

- General cost function that we consider:

$$J_{WLS}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).$$

- For some models, closed form solutions do exist – but for most they do not.
- Regularized cost functions can be handled with an augmentation trick – we will come back to that.

Nonlinear Model of an Autonomous Car

- We measure the range to each landmark:

$$y_1^R = \sqrt{(s_1^x - p^x)^2 + (s_1^y - p^y)^2} + r_1^R,$$

⋮

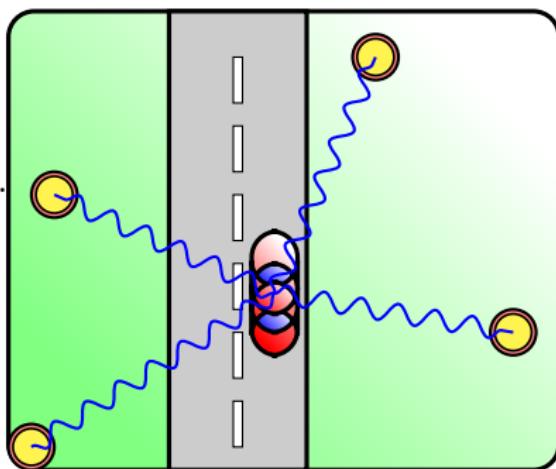
$$y_M^R = \sqrt{(s_M^x - p^x)^2 + (s_M^y - p^y)^2} + r_M^R.$$

- This is a non-linear model

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r}$$

- We can find $\mathbf{x} = (p^x, p^y)$ by minimizing the cost function

$$J_{WLS}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).$$



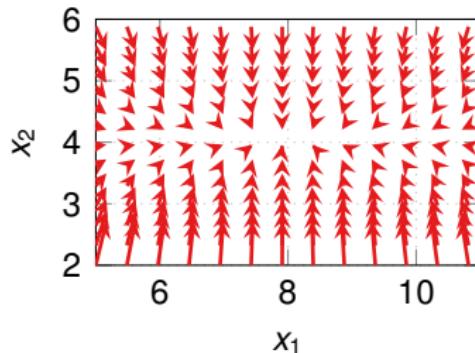
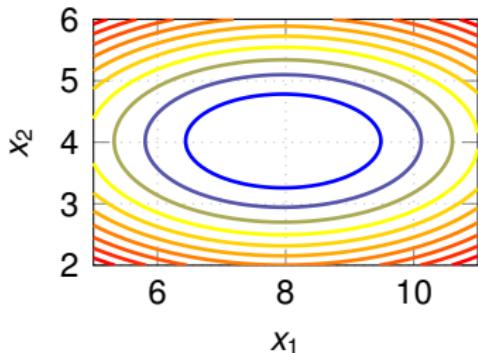
Numerical Optimization

- Iterative algorithms can be used to find the minima of a cost function.
- Generally find local minima \Rightarrow require good initialization.
- Today, we will look at two approaches:
 - 1 Gradient descent method
 - 2 the Gauss–Newton algorithm

Gradient Descent: Formulation

- The gradient of $J(\mathbf{x})$ w.r.t. \mathbf{x} points to the direction where $J(\mathbf{x})$ increases as a function of \mathbf{x}
- Changing \mathbf{x} in the opposite direction of the gradient decreases $J(\mathbf{x})$
- If the function to minimize is $J_{WLS}(\mathbf{x})$, the cost is decreased by the iteration

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} - \gamma \left. \frac{\partial J_{WLS}(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\hat{\mathbf{x}}^{(i)}} ,$$



Gradient Descent: Derivation (1/3)

- Let us start by considering scalar LS cost function

$$J_{\text{LS}}(\mathbf{x}) = \sum_{n=1}^N (y_n - g_n(\mathbf{x}))^2$$

- The gradient is

$$\begin{aligned}\frac{\partial J_{\text{LS}}(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial}{\partial \mathbf{x}} \sum_{n=1}^N (y_n - g_n(\mathbf{x}))^2 \\ &= \sum_{n=1}^N -2(y_n - g_n(\mathbf{x})) \frac{\partial g_n(\mathbf{x})}{\partial \mathbf{x}}.\end{aligned}$$

Gradient Descent: Derivation (2/3)

- Vector form:

$$\frac{\partial J_{\text{LS}}(\mathbf{x})}{\partial \mathbf{x}} = \sum_{n=1}^N -2(y_n - g_n(\mathbf{x})) \frac{\partial g_n(\mathbf{x})}{\partial \mathbf{x}}$$

$$= -2 \begin{bmatrix} \frac{\partial g_1(\mathbf{x})}{\partial \mathbf{x}} & \frac{\partial g_2(\mathbf{x})}{\partial \mathbf{x}} & \dots & \frac{\partial g_N(\mathbf{x})}{\partial \mathbf{x}} \end{bmatrix} \left(\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_N(\mathbf{x}) \end{bmatrix} \right)$$

$$= -2 \begin{bmatrix} \frac{\partial g_1(\mathbf{x})}{\partial x_1} & \frac{\partial g_2(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial g_N(\mathbf{x})}{\partial x_1} \\ \frac{\partial g_1(\mathbf{x})}{\partial x_2} & \frac{\partial g_2(\mathbf{x})}{\partial x_2} & & \vdots \\ \vdots & & \ddots & \frac{\partial g_N(\mathbf{x})}{\partial x_{K-1}} \\ \frac{\partial g_1(\mathbf{x})}{\partial x_K} & \dots & \frac{\partial g_{N-1}(\mathbf{x})}{\partial x_K} & \frac{\partial g_N(\mathbf{x})}{\partial x_K} \end{bmatrix} (\mathbf{y} - \mathbf{g}(\mathbf{x}))$$

$$= -2 \mathbf{G}_{\mathbf{x}}^T(\mathbf{x})(\mathbf{y} - \mathbf{g}(\mathbf{x})).$$

- $\mathbf{G}_{\mathbf{x}}$ is the Jacobian matrix of $\mathbf{g}(\mathbf{x})$

Gradient Descent: Derivation (3/3)

- Generalization to WLS cost function:

$$\begin{aligned}\frac{\partial J_{\text{WLS}}(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial}{\partial \mathbf{x}} (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})) \\ &= -2 \mathbf{G}_x^\top(\mathbf{x}) \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).\end{aligned}$$

- The direction of negative gradient is $\mathbf{G}_x^\top(\mathbf{x}) \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x}))$.
- The parameter update becomes:

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma \mathbf{G}_x^\top(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).$$

- We have absorbed the factor 2 into the constant γ .

Gradient Descent: Step Size

- How long the step length γ should be chosen?
 - Choosing γ too large may cause the cost function to increase.
 - Too small steps might cause unnecessarily slow convergence.
- A typical strategy is to simply choose it small enough so that the cost decreases at every step.
- Advisable to change the step length during the iterations in one way or another.
- One way is to use a line search – but we come back to that next time.

Gradient Descent: Algorithm

Algorithm 1 Gradient Descent

Require: Initial parameter guess $\hat{\mathbf{x}}^{(0)}$, data \mathbf{y} , function $\mathbf{g}(\mathbf{x})$, Jacobian $\mathbf{G}_{\mathbf{x}}(\mathbf{x})$

Ensure: Parameter estimate $\hat{\mathbf{x}}_{WLS}$

- 1: Set $i \leftarrow 0$
- 2: **repeat**
- 3: Calculate the update direction

$$\Delta \mathbf{x}^{(i+1)} = \mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

- 4: Select a suitable $\gamma^{(i+1)}$
- 5: Calculate

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma^{(i+1)} \Delta \mathbf{x}^{(i+1)}$$

- 6: Set $i \leftarrow i + 1$
 - 7: **until** Converged
-

Example: Localizing a Car (1)

- We have

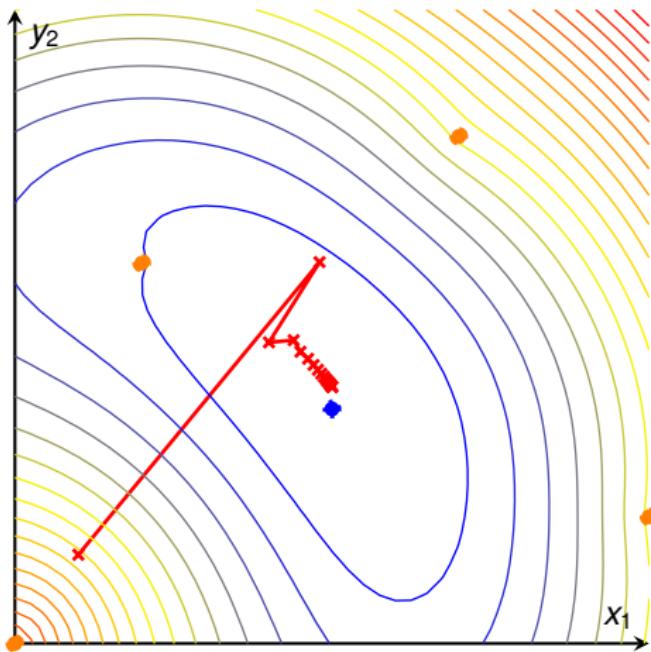
$$y_n^R = \underbrace{\sqrt{(s_n^x - p^x)^2 + (s_n^y - p^y)^2}}_{g_n(\mathbf{x})} + r_n^R, \quad \mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ \vdots \\ g_M(\mathbf{x}) \end{bmatrix}$$

- The Jacobian is

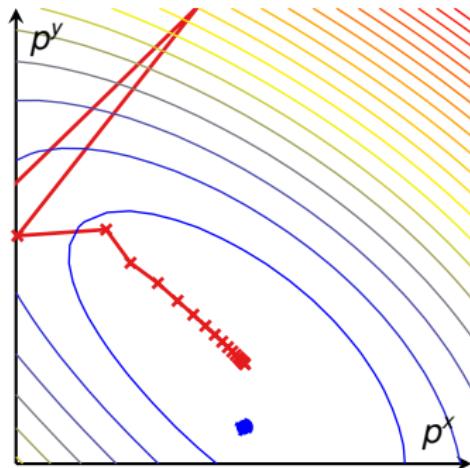
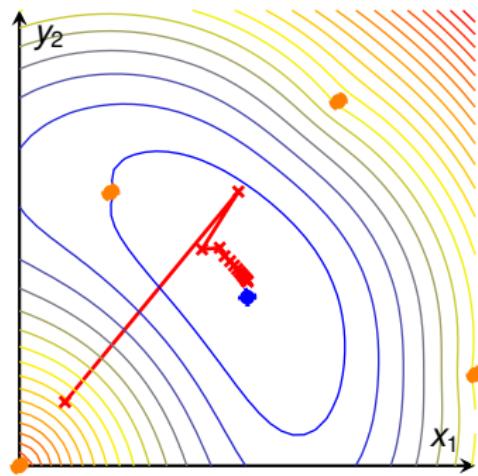
$$\mathbf{G}_{\mathbf{x}}(\mathbf{x}) = \begin{bmatrix} \frac{-(s_1^x - p^x)}{\sqrt{(s_1^x - p^x)^2 + (s_1^y - p^y)^2}} & \frac{-(s_1^y - p^y)}{\sqrt{(s_1^x - p^x)^2 + (s_1^y - p^y)^2}} \\ \vdots & \vdots \\ \frac{-(s_M^x - p^x)}{\sqrt{(s_M^x - p^x)^2 + (s_M^y - p^y)^2}} & \frac{-(s_M^y - p^y)}{\sqrt{(s_M^x - p^x)^2 + (s_M^y - p^y)^2}} \end{bmatrix}$$

- Tip:* always, **always** check the Jacobian numerically!

Example: Localizing a Car (2)



Example: Localizing a Car (3)



Gauss–Newton Algorithm: Derivation (1/2)

- Idea: Given $\hat{\mathbf{x}}^{(i)}$, we can linearize the nonlinear measurement model around that point
- Linearized measurement model:

$$\mathbf{g}(\mathbf{x}) \approx \mathbf{g}(\hat{\mathbf{x}}^{(i)}) + \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) (\mathbf{x} - \hat{\mathbf{x}}^{(i)})$$

- Cost function approximation:

$$\begin{aligned} J_{\text{WLS}}(\mathbf{x}) &\approx \left(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}) - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) (\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \right)^T \mathbf{R}^{-1} \\ &\quad \times \left(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}) - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) (\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \right) \\ &= \left(\mathbf{e}^{(i)} - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) (\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \right)^T \mathbf{R}^{-1} \\ &\quad \times \left(\mathbf{e}^{(i)} - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) (\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \right) \end{aligned}$$

- This can now be minimized w.r.t. \mathbf{x} in the same way as linear models.

Gauss–Newton Algorithm: Derivation (2/2)

- Gradient of the cost function approximation w.r.t. \mathbf{x} :

$$\begin{aligned}\frac{\partial J_{\text{WLS}}(\mathbf{x})}{\partial \mathbf{x}} &\approx \frac{\partial}{\partial \mathbf{x}} \left((\mathbf{e}^{(i)})^\top \mathbf{R}^{-1} \mathbf{e}^{(i)} - (\mathbf{e}^{(i)})^\top \mathbf{R}^{-1} \mathbf{G}_x(\hat{\mathbf{x}}^{(i)}) (\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \right. \\ &\quad - (\mathbf{x} - \hat{\mathbf{x}}^{(i)})^\top \mathbf{G}_x^\top(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{e}^{(i)} \\ &\quad \left. + (\mathbf{x} - \hat{\mathbf{x}}^{(i)})^\top \mathbf{G}_x^\top(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_x(\hat{\mathbf{x}}^{(i)}) (\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \right) \\ &= -2 \mathbf{G}_x^\top(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{e}^{(i)} + 2 \mathbf{G}_x^\top(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_x(\hat{\mathbf{x}}^{(i)}) (\mathbf{x} - \hat{\mathbf{x}}^{(i)})\end{aligned}$$

- Setting to zero and solving for \mathbf{x} gives:

$$\begin{aligned}\mathbf{x} &= \hat{\mathbf{x}}^{(i)} + (\mathbf{G}_x^\top(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_x(\hat{\mathbf{x}}^{(i)}))^{-1} \mathbf{G}_x^\top(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{e}^{(i)} \\ &= \hat{\mathbf{x}}^{(i)} + (\mathbf{G}_x^\top(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_x(\hat{\mathbf{x}}^{(i)}))^{-1} \mathbf{G}_x^\top(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).\end{aligned}$$

- The Gauss–Newton method we used the above solution \mathbf{x} as the next iterate $\hat{\mathbf{x}}^{(i+1)}$.

Gauss–Newton Algorithm: Algorithm

Algorithm 2 Gauss–Newton Algorithm

Require: Initial parameter guess $\hat{\mathbf{x}}^{(0)}$, data \mathbf{y} , function $\mathbf{g}(\mathbf{x})$, Jacobian \mathbf{G}_x

Ensure: Parameter estimate $\hat{\mathbf{x}}_{WLS}$

- 1: Set $i \leftarrow 0$
- 2: **repeat**
- 3: Calculate the update direction

$$\Delta \mathbf{x}^{(i+1)} = (\mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_x(\hat{\mathbf{x}}^{(i)}))^{-1}\mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

- 4: Calculate

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \Delta \mathbf{x}^{(i+1)}$$

- 5: Set $i \leftarrow i + 1$
 - 6: **until** Converged
 - 7: Set $\hat{\mathbf{x}}_{WLS} = \hat{\mathbf{x}}^{(i)}$
-

Gauss–Newton Algorithm: Covariance of the Estimate

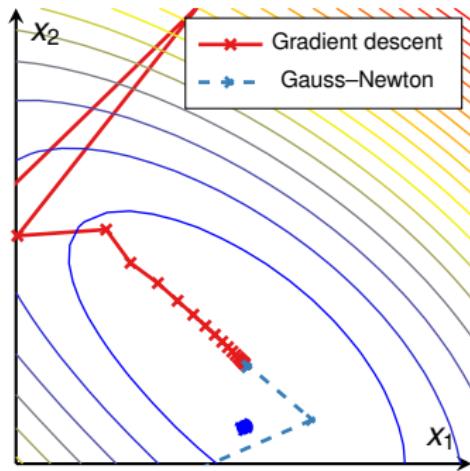
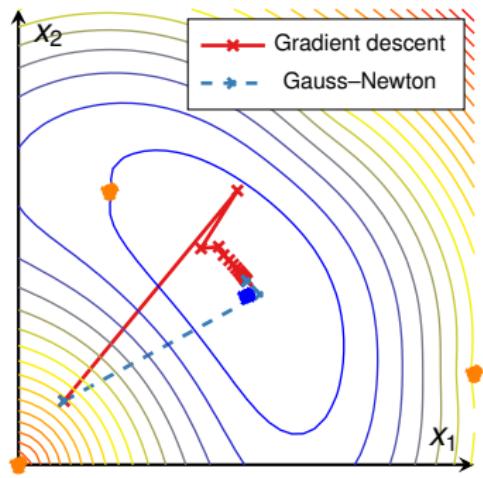
- The covariance of the estimate is hard to compute in the non-linear case.
- However, we can use the linearization approximation:

$$\mathbf{g}(\mathbf{x}) \approx \mathbf{g}(\hat{\mathbf{x}}_{WLS}) + \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}_{WLS})(\mathbf{x} - \hat{\mathbf{x}}_{WLS})$$

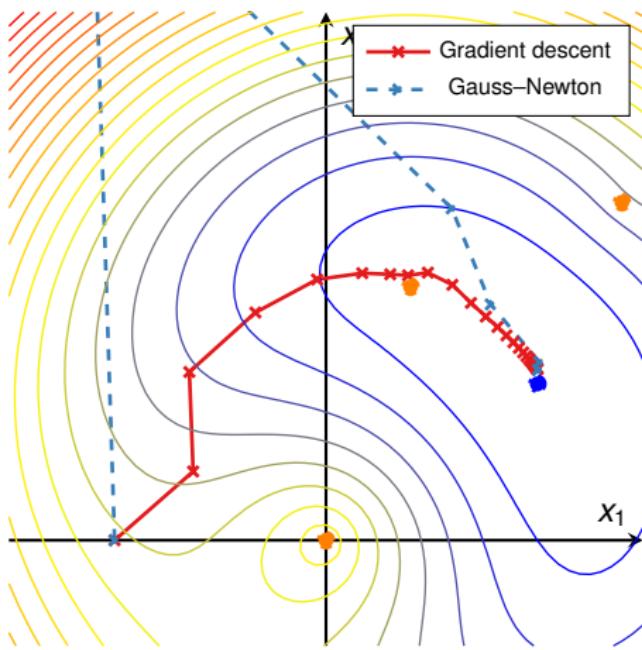
- The covariance for this linear model can be used as approximation for the covariance:

$$\text{Cov}\{\hat{\mathbf{x}}_{WLS}\} \approx (\mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}_{WLS}) \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}_{WLS}))^{-1}.$$

Example: Localizing a Car (4)



Example: Localizing a Car (5)



Gauss–Newton Algorithm: Challenges

- When the initial point is far away, the convergence can fail.
 - ✓ We can try several starting points or select them cleverly.
- The full step using the linearized model might go too far:
 - ✓ Line search can be used to select a good step length.
 - ✓ We could use regularized solution for the linearized model.
- The latter two methods will be presented next time.

Summary

- Sensor fusion problems are often **nonlinear**.
- General nonlinear model has the form:

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r},$$

- General cost function that we considered:

$$J_{\text{WLS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).$$

- Gradient descent algorithm takes steps towards the direction of negative gradient.
- Gauss–Newton iteratively linearizes the model and solves the linear optimization problem.



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Gauss-Newton with Line Search and Levenberg-Marquardt Algorithm

Simo Särkkä

Aalto University

October 9, 2020

Contents

- 1 Intended Learning Outcomes and Recap
- 2 Gauss–Newton with Line Search
- 3 Levenberg–Marquardt Algorithm
- 4 Regularized Problems, Convergence Criteria, and Quasi–Newton
- 5 Summary

Intended Learning Outcomes

After this lecture, you will be able to:

- **understand** the principle of line search in Gauss–Newton method;
- **understand** the principle of Levenberg–Marquardt algorithm;
- **apply** Gauss–Newton method with line search and Levenberg–Marquardt algorithm to sensor fusion problems.

Recap (1)

- Sensor fusion problems are often **nonlinear**.
- General nonlinear model has the form:

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r},$$

- General cost function that we considered:

$$J_{\text{WLS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).$$

- Gradient descent algorithm takes steps towards the direction of negative gradient.
- Gauss–Newton iteratively linearizes the model and solves the linear optimization problem.

Recap (2)

Algorithm 1 Gradient Descent

Require: Initial parameter guess $\hat{\mathbf{x}}^{(0)}$, data \mathbf{y} , function $\mathbf{g}(\mathbf{x})$, Jacobian $\mathbf{G}_{\mathbf{x}}(\mathbf{x})$

Ensure: Parameter estimate $\hat{\mathbf{x}}_{WLS}$

- 1: Set $i \leftarrow 0$
- 2: **repeat**
- 3: Calculate the update direction

$$\Delta \mathbf{x}^{(i+1)} = \mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

- 4: Select a suitable $\gamma^{(i+1)}$
- 5: Calculate

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma^{(i+1)} \Delta \mathbf{x}^{(i+1)}$$

- 6: Set $i \leftarrow i + 1$
 - 7: **until** Converged
-

Recap (3)

Algorithm 2 Gauss–Newton Algorithm

Require: Initial parameter guess $\hat{\mathbf{x}}^{(0)}$, data \mathbf{y} , function $\mathbf{g}(\mathbf{x})$, Jacobian $\mathbf{G}_{\mathbf{x}}$

Ensure: Parameter estimate $\hat{\mathbf{x}}_{WLS}$

- 1: Set $i \leftarrow 0$
- 2: **repeat**
- 3: Calculate the update direction

$$\Delta \mathbf{x}^{(i+1)} = (\mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}))^{-1} \mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

- 4: Calculate

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \Delta \mathbf{x}^{(i+1)}$$

- 5: Set $i \leftarrow i + 1$
 - 6: **until** Converged
 - 7: Set $\hat{\mathbf{x}}_{WLS} = \hat{\mathbf{x}}^{(i)}$
-

Gauss–Newton Algorithm with Line Search: Motivation

- Gauss–Newton uses linearization to determine the next iterate.
- As linearization is a local approximation, taking the full step might over/undershoot.
- Instead, we can use scaled Gauss–Newton step:

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma \Delta \hat{\mathbf{x}}^{(i+1)},$$

$$\Delta \hat{\mathbf{x}}^{(i+1)} = (\mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_x(\hat{\mathbf{x}}^{(i)}))^{-1} \mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})),$$

- Here γ is the scaling factor – typically $\gamma \in [0, 1]$.
- How should we select the scaling factor?

Gauss–Newton Algorithm with Line Search: Derivation

- One way to select the scaling parameter γ is to use a line search.
- Given the Gauss–Newton increment $\Delta\hat{\mathbf{x}}^{(i+1)}$, we can consider cost as function of the scale parameter:

$$J_{\text{WLS}}^{(i)}(\gamma) = J_{\text{WLS}} \left(\hat{\mathbf{x}}^{(i)} + \gamma \Delta\hat{\mathbf{x}}^{(i+1)} \right).$$

- Then the idea of line search is to locally optimize the above function in the range $[0, 1]$.
- In the exact line search we simply find the minimum e.g. by evaluating it in grid.
- We could also use bisection algorithm or interpolation methods to find the minimum.

Exact Line Search on Grid

Algorithm 3 Line Search on Grid

Require: Previous iterate $\hat{\mathbf{x}}^{(i)}$, the update direction $\Delta\hat{\mathbf{x}}^{(i+1)}$, the cost function $J_{WLS}(\mathbf{x})$, and the grid size N_g .

Ensure: Optimal step size γ^* .

- 1: Set $\gamma^* \leftarrow 0$ and $J^* \leftarrow J_{WLS}(\hat{\mathbf{x}}^{(i)})$
 - 2: **for** $j \in \{1, 2, \dots, N_g\}$ **do**
 - 3: Set $\gamma \leftarrow j/N_g$
 - 4: **if** $J_{WLS}(\hat{\mathbf{x}}^{(i)} + \gamma\Delta\hat{\mathbf{x}}^{(i+1)}) < J^*$ **then**
 - 5: Set $\gamma^* \leftarrow \gamma$
 - 6: **end if**
 - 7: **end for**
-

Inexact Line Search (1/2)

- The line search does **not need to be exact** to guarantee to find the minimum.
- In **backtracking** we decrease the parameter γ until it provides a sufficient decrease in the cost.
- One way is to **halve the step size** until the cost decreases.
- In **Armijo backtracking** we demand that the cost is decreased at least with an amount that is predicted by a first order Taylor series expansion.
- The **first order Taylor series expansion** for the cost as function of scale parameter gives:

$$\begin{aligned} J_{WLS} & \left(\hat{\mathbf{x}}^{(i)} + \gamma \Delta \hat{\mathbf{x}}^{(i+1)} \right) \\ & \approx J_{WLS} \left(\hat{\mathbf{x}}^{(i)} \right) - 2\gamma [\Delta \hat{\mathbf{x}}^{(i+1)}]^T \mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)}) (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})). \end{aligned}$$

Inexact Line Search (2/2)

- Then we demand that the cost decrease should satisfy the **Armijo condition**

$$\begin{aligned} J_{WLS}\left(\hat{\mathbf{x}}^{(i)} + \gamma \Delta \hat{\mathbf{x}}^{(i+1)}\right) - J_{WLS}\left(\hat{\mathbf{x}}^{(i)}\right) \\ \leq -2\beta \gamma [\Delta \hat{\mathbf{x}}^{(i+1)}]^T \mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})). \end{aligned}$$

- Here β is a **parameter** that we can choose freely on range $[0, 1]$ (e.g. $\beta = 0.1$).
- We then decrease γ by **multiplying it with a parameter τ** (e.g., $\tau = 0.5$) on the range $(0, 1)$ until the condition is satisfied:

$$\gamma \leftarrow \tau \gamma.$$

Line Search with Armijo Backtracking

Algorithm 4 Line Search with Armijo Backtracking

Require: Previous iterate $\hat{\mathbf{x}}^{(i)}$, the update direction $\Delta\hat{\mathbf{x}}^{(i+1)}$, the cost function $J_{WLS}(\mathbf{x})$, and parameters β and τ .

Ensure: Suitable step size γ .

- 1: Set $\gamma \leftarrow 1$ and $J_0 \leftarrow J_{WLS}(\hat{\mathbf{x}}^{(i)})$.
 - 2: Set $d \leftarrow -2\beta [\Delta\hat{\mathbf{x}}^{(i+1)}]^T \mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)}) (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$
 - 3: **while** $J_{WLS}(\hat{\mathbf{x}}^{(i)} + \gamma\Delta\hat{\mathbf{x}}^{(i+1)}) > J_0 + \gamma d$ **do**
 - 4: Set $\gamma \leftarrow \tau \gamma$
 - 5: **end while**
-

Gauss–Newton Algorithm with Line Search

Algorithm 5 Gauss–Newton Algorithm with Line Search

Require: Initial parameter guess $\hat{\mathbf{x}}^{(0)}$, data \mathbf{y} , function $\mathbf{g}(\mathbf{x})$, Jacobian $\mathbf{G}_{\mathbf{x}}(\mathbf{x})$

Ensure: Parameter estimate $\hat{\mathbf{x}}_{WLS}$

1: Set $i \leftarrow 0$

2: **repeat**

3: Calculate the update direction

$$\Delta \mathbf{x}^{(i+1)} = (\mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}))^{-1} \mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

4: Compute optimal $\gamma^{(i+1)}$ with line search (Algorithm 3 or 4)

5: Calculate

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma^{(i+1)} \Delta \mathbf{x}^{(i+1)}$$

6: Set $i \leftarrow i + 1$

7: **until** Converged

8: Set $\hat{\mathbf{x}}_{WLS} = \hat{\mathbf{x}}^{(i)}$

Gauss–Newton Algorithm with Line Search: Example (1/3)

- We measure the range to each landmark:

$$y_1^R = \sqrt{(s_1^x - p^x)^2 + (s_1^y - p^y)^2} + r_1^R,$$

⋮

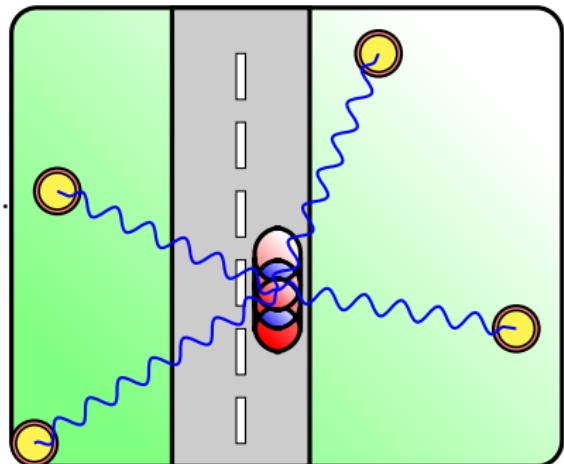
$$y_M^R = \sqrt{(s_M^x - p^x)^2 + (s_M^y - p^y)^2} + r_M^R.$$

- This is a non-linear model

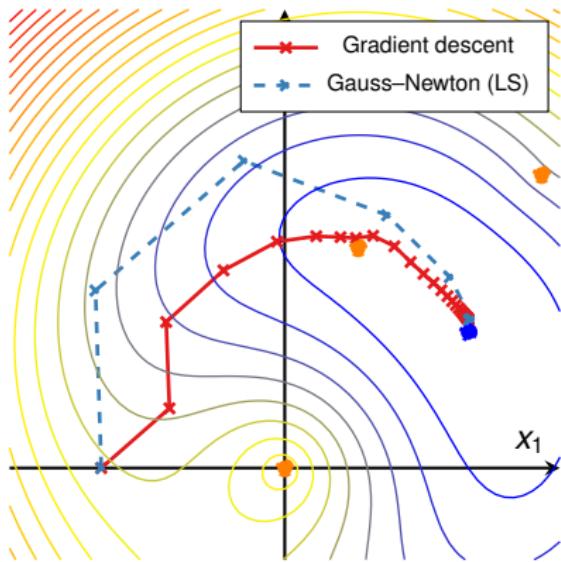
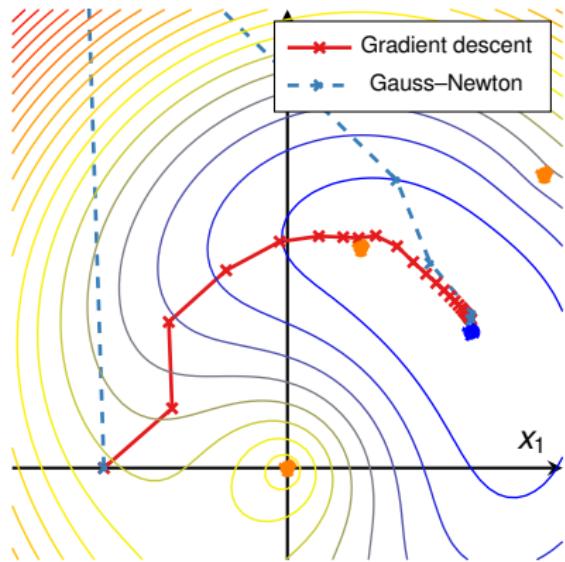
$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{r}$$

- We can find $\mathbf{x} = (p^x, p^y)$ by minimizing the cost function

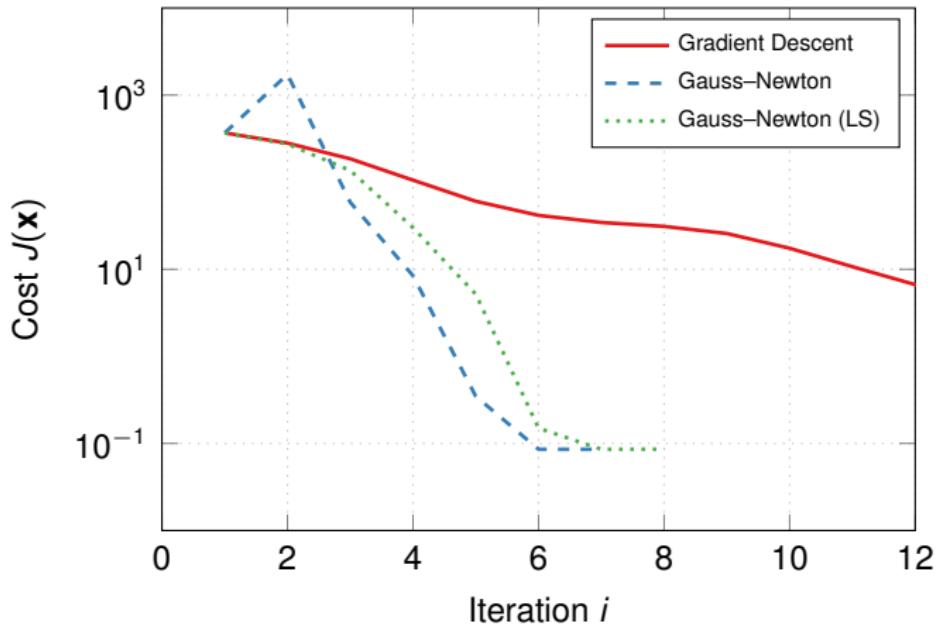
$$J_{WLS}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).$$



Gauss–Newton Algorithm with Line Search: Example (2/3)



Gauss–Newton Algorithm with Line Search: Example (3/3)



Levenberg–Marquardt Algorithm: Motivation

- Gradient descent: Quickly moves to low cost area, creeps to minimum
- Gauss–Newton: Straight to minimum, may take a detour
- Can we have the best of both worlds?
- ... kind of, the Levenberg–Marquardt algorithm.

Levenberg–Marquardt Algorithm: Derivation

- The Levenberg–Marquardt algorithm can be seen as a regularized version of the Gauss–Newton algorithm.
- Cost function approximation:

$$\begin{aligned} J_{\text{ReLS}}(\mathbf{x}) \approx & \left(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}) - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})(\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \right)^T \\ & \times \mathbf{R}^{-1} \left(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}) - \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})(\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \right) \\ & + \lambda (\mathbf{x} - \hat{\mathbf{x}}^{(i)})^T (\mathbf{x} - \hat{\mathbf{x}}^{(i)}) \end{aligned}$$

- We can now minimize this as a linear regularized problem:

$$\mathbf{x} = \hat{\mathbf{x}}^{(i)} + (\mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) + \lambda \mathbf{I})^{-1} \mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

- Using this as the next iterate gives the iteration:

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \Delta \hat{\mathbf{x}}^{(i+1)},$$

$$\Delta \hat{\mathbf{x}}^{(i+1)} = (\mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) + \lambda \mathbf{I})^{-1} \mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).$$

Levenberg–Marquardt Algorithm: Damping (1/2)

- The Levenberg–Marquardt update:

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \Delta\hat{\mathbf{x}}^{(i+1)},$$

$$\Delta\hat{\mathbf{x}}^{(i+1)} = (\mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_x(\hat{\mathbf{x}}^{(i)}) + \lambda\mathbf{I})^{-1}\mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).$$

- How should the damping parameter λ be chosen?
- If $\lambda \rightarrow 0$:

$$\Delta\hat{\mathbf{x}}^{(i+1)} \rightarrow (\mathbf{G}_x^T\mathbf{R}^{-1}\mathbf{G}_x)^{-1}\mathbf{G}_x^T\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$

- If $\lambda \rightarrow \infty$:

$$\Delta\hat{\mathbf{x}}^{(i+1)} \rightarrow \frac{1}{\lambda}\mathbf{G}_x^T\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).$$

- Ideally, in flat regions of the cost function where the linear approximation is good, λ should be chosen small, whereas in steep regions, it should be chosen large.

Levenberg–Marquardt Algorithm: Damping (2/2)

- A simple strategy is to start from some damping value $\lambda^{(0)}$ (e.g. $\lambda^{(0)} = 10^{-2}$) and select a fixed factor ν (e.g. $\nu = 10$).
- Then at each step we do the following:
 - First compute a candidate $\hat{\mathbf{x}}^{(i+1)}$ using the previous parameter value $\lambda^{(i-1)}$. Then proceed as follows:
 - If $J_{WLS}(\hat{\mathbf{x}}^{(i+1)}) < J_{WLS}(\hat{\mathbf{x}}^{(i)})$ then accept $\hat{\mathbf{x}}^{(i+1)}$ and decrease the damping parameter by $\lambda^{(i)} = \lambda^{(i-1)}/\nu$.
 - Otherwise continue with $\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)}$ and increase the damping parameter by $\lambda^{(i)} = \nu \lambda^{(i-1)}$.
- This idea appears already in the original article of Marquadt (1963).
- More sophisticated adaptation schemes can also be found in literature.

Levenberg–Marquardt Algorithm: Scaling

- The resulting algorithm is not scale invariant.
- To make it scale invariant we can normalize the regularized cost function approximation by using the diagonal values of $\mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_x(\hat{\mathbf{x}}^{(i)})$.
- This is equivalent to replacing the regularization term $\lambda \mathbf{I}$ with $\lambda \text{diag}(\mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_x(\hat{\mathbf{x}}^{(i)}))$.
- The scaled iteration then becomes:

$$\begin{aligned}\hat{\mathbf{x}}^{(i+1)} &= \hat{\mathbf{x}}^{(i)} + \Delta\hat{\mathbf{x}}^{(i+1)}, \\ \Delta\hat{\mathbf{x}}^{(i+1)} &= (\mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_x(\hat{\mathbf{x}}^{(i)}) + \lambda \text{diag}(\mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}\mathbf{G}_x(\hat{\mathbf{x}}^{(i)})))^{-1} \\ &\quad \times \mathbf{G}_x^T(\hat{\mathbf{x}}^{(i)})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)})).\end{aligned}$$

Levenberg–Marquardt Algorithm: Algorithm

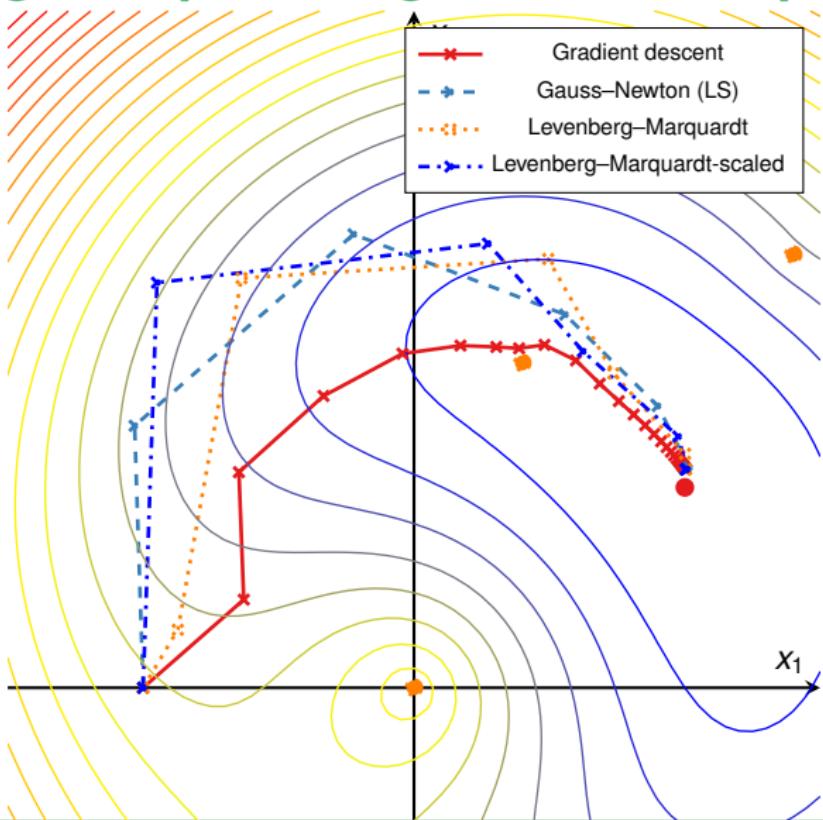
Algorithm 4.6 Levenberg–Marquardt Algorithm with simple adaptation

Input: Initial parameter guess $\hat{\mathbf{x}}^{(0)}$, data \mathbf{y} , function $\mathbf{g}(\mathbf{x})$, Jacobian $\mathbf{G}_{\mathbf{x}}$, initial damping $\lambda^{(0)}$, and parameter ν .

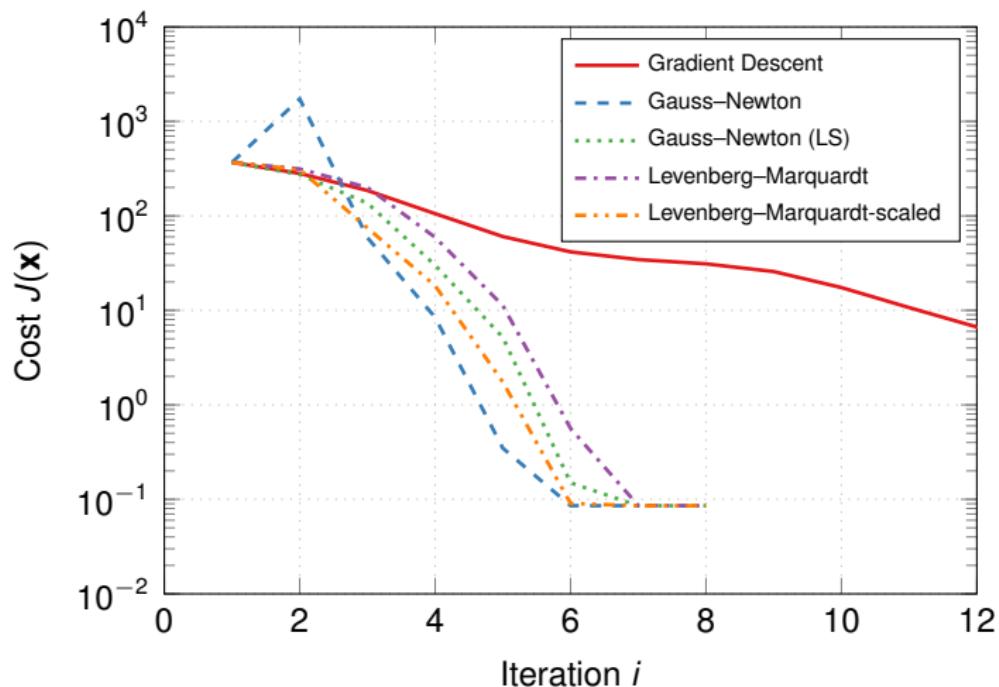
Output: Parameter estimate $\hat{\mathbf{x}}_{WLS}$

- 1: Set $i \leftarrow 0$ and $\lambda \leftarrow \lambda^{(0)}$.
 - 2: **repeat**
 - 3: Compute the (candidate) parameter update:
 - 4: **if** using scaled version **then**
$$\Delta \hat{\mathbf{x}}^{(i+1)} = (\mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) + \lambda \text{diag}(\mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)})))^{-1} \times \mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$
 - 5: **else**
$$\Delta \hat{\mathbf{x}}^{(i+1)} = (\mathbf{G}_{\mathbf{x}}^T(\hat{\mathbf{x}}^{(i)}) \mathbf{R}^{-1} \mathbf{G}_{\mathbf{x}}(\hat{\mathbf{x}}^{(i)}) + \lambda \mathbf{I})^{-1} \mathbf{G}_{\mathbf{x}}^T \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\hat{\mathbf{x}}^{(i)}))$$
 - 6: **end if**
 - 7: **if** $J_{WLS}(\hat{\mathbf{x}}^{(i)} + \Delta \hat{\mathbf{x}}^{(i+1)}) < J_{WLS}(\hat{\mathbf{x}}^{(i)})$ **then**
 - 8: Accept the candidate and decrease λ :
$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \Delta \hat{\mathbf{x}}^{(i+1)}$$
$$\lambda \leftarrow \lambda / \nu$$
 - 9: Set $i \leftarrow i + 1$
 - 10: **else**
 - 11: Reject the candidate and increase λ :
$$\lambda \leftarrow \nu \lambda$$
 - 12: **end if**
 - 13: **until** Converged
 - 14: Set $\hat{\mathbf{x}}_{WLS} = \hat{\mathbf{x}}^{(i)}$
-

Levenberg–Marquardt Algorithm: Example



Decrease in Cost



Regularized Non-Linear Models

- The cost function that we considered:

$$J_{\text{WLS}}(\mathbf{x}) = (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})).$$

- However, sometimes we are interested in regularized cost functions.
- Luckily, we can use the following simple trick:

$$\begin{aligned} J_{\text{ReLS}}(\mathbf{x}) &= (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})) + (\mathbf{m} - \mathbf{x})^\top \mathbf{P}^{-1} (\mathbf{m} - \mathbf{x}) \\ &= \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \right)^\top \begin{bmatrix} \mathbf{R}^{-1} & 0 \\ 0 & \mathbf{P}^{-1} \end{bmatrix} \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \right). \end{aligned}$$

- This is now a non-regularized cost function and hence all the algorithms presented are applicable.

Quasi-Newton Methods (1/2)

- Here we have only concentrated on least squares problems.
- There also exists a wider class of quasi-Newton methods.
- Let us consider a generic cost function $J(\mathbf{x})$ which we wish to minimize.
- Assume that our current guess for the minimum is $\mathbf{x}^{(i)}$ – we can now Taylor expand the cost function as follows:

$$\begin{aligned} J(\mathbf{x}) \approx J(\mathbf{x}^{(i)}) &+ \left[\frac{\partial J(\mathbf{x})}{\partial \mathbf{x}} \right]^\top \Bigg|_{\mathbf{x}=\mathbf{x}^{(i)}} (\mathbf{x} - \mathbf{x}^{(i)}) \\ &+ \frac{1}{2} (\mathbf{x} - \mathbf{x}^{(i)})^\top \left[\frac{\partial^2 J(\mathbf{x})}{\partial \mathbf{x}^2} \right] \Bigg|_{\mathbf{x}=\mathbf{x}^{(i)}} (\mathbf{x} - \mathbf{x}^{(i)}). \end{aligned}$$

- We can now minimize the right hand side with respect to \mathbf{x} and use the result as the next guess.

Quasi-Newton Methods (2/2)

- The resulting Newton's method takes the following form:

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \left[\frac{\partial^2 J(\mathbf{x})}{\partial \mathbf{x}^2} \right]^{-1} \left. \frac{\partial J(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}^{(i)}}.$$

- However, computation of the Hessian is often not desirable.
- In so called quasi-Newton methods the Hessian is approximated in various ways.
- The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is a famous method for this.
- Gauss–Newton method can also be seen as a quasi–Newton method where we approximate the Hessian by $2\mathbf{G}_x^T \mathbf{R}^{-1} \mathbf{G}_x$.
- The line search procedure is typically an essential part of quasi-Newton methods.

Convergence Criteria

- When should we terminate iterations in optimization method?
- Various criteria and their combinations are used:
 - The absolute or relative change in the parameter estimate falls below a threshold, e.g.:

$$\|\Delta \mathbf{x}^{(i)}\| < \epsilon_p$$

- The absolute or relative change in the cost falls below a certain threshold, e.g.:

$$\left| \frac{(J(\mathbf{x}^i) - J(\mathbf{x}^{(i+1)}))}{J(\mathbf{x}^i)} \right| < \epsilon_c$$

- A maximum number of iterations is reached.

Summary (1/2)

- The Gauss–Newton update can be scaled with additional parameter γ :

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma \Delta \hat{\mathbf{x}}^{(i+1)}.$$

- The parameter can be found via line search that minimizes

$$J_{\text{WLS}}^{(i)}(\gamma) = J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)} + \gamma \Delta \hat{\mathbf{x}}^{(i+1)}\right).$$

- We can also use inexact line search which ensures that the cost is decreased a sufficient amount.
- In Levenberg–Marquardt (LM) algorithm we replace the linear approximation in Gauss–Newton with its regularized version.
- In LM algorithm, we find a suitable regularization parameter λ via an iterative procedure.

Summary (2/2)

- We can also consider regularized nonlinear problems with a simple trick:

$$\begin{aligned} J_{\text{ReLS}}(\mathbf{x}) &= (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})) + (\mathbf{m} - \mathbf{x})^\top \mathbf{P}^{-1} (\mathbf{m} - \mathbf{x}) \\ &= \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \right)^\top \begin{bmatrix} \mathbf{R}^{-1} & 0 \\ 0 & \mathbf{P}^{-1} \end{bmatrix} \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \right). \end{aligned}$$

- Quasi-Newton methods are more general optimization methods that approximate the Hessian in Newton's method.
- Various convergence criteria are available for terminating iterative optimization methods.



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Continuous-Time Dynamic Models

Simo Särkkä

Aalto University

October 16, 2020

Contents

- 1 Intended Learning Outcomes and Recap
- 2 Motivation: Need for Dynamic Models
- 3 Linear ODEs and State-Space Models
- 4 Stochastic Modeling of Unknown Components
- 5 Nonlinear State-Space Models

Intended Learning Outcomes

After this lecture, you will be able to:

- describe the idea of dynamic modeling in sensor fusion,
- explain the process of constructing continuous-time state-space models,
- distinguish deterministic and stochastic state-space models,
- construct linear and nonlinear continuous-time state-space models.

Recap (1/2)

- The Gauss–Newton update can be scaled with additional parameter γ :

$$\hat{\mathbf{x}}^{(i+1)} = \hat{\mathbf{x}}^{(i)} + \gamma \Delta \hat{\mathbf{x}}^{(i+1)}.$$

- The parameter can be found via line search that minimizes

$$J_{\text{WLS}}^{(i)}(\gamma) = J_{\text{WLS}}\left(\hat{\mathbf{x}}^{(i)} + \gamma \Delta \hat{\mathbf{x}}^{(i+1)}\right).$$

- We can also use inexact line search which ensures that the cost is decreased a sufficient amount.
- In Levenberg–Marquardt (LM) algorithm we replace the linear approximation in Gauss–Newton with its regularized version.
- In LM algorithm, we find a suitable regularization parameter λ via an iterative procedure.

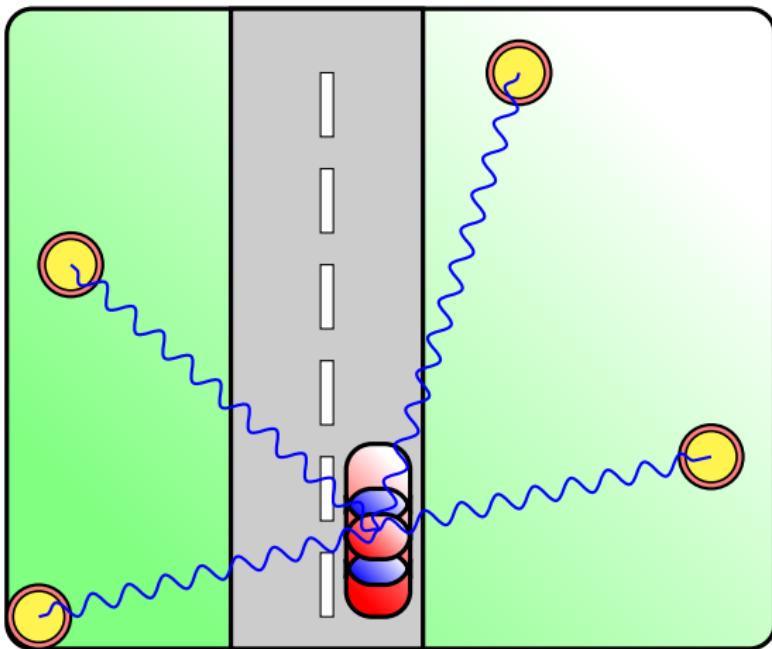
Recap (2/2)

- We can also consider **regularized nonlinear problems** with a simple trick:

$$\begin{aligned} J_{\text{ReLS}}(\mathbf{x}) &= (\mathbf{y} - \mathbf{g}(\mathbf{x}))^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{x})) + (\mathbf{m} - \mathbf{x})^\top \mathbf{P}^{-1} (\mathbf{m} - \mathbf{x}) \\ &= \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \right)^\top \begin{bmatrix} \mathbf{R}^{-1} & 0 \\ 0 & \mathbf{P}^{-1} \end{bmatrix} \left(\begin{bmatrix} \mathbf{y} \\ \mathbf{m} \end{bmatrix} - \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \right). \end{aligned}$$

- Quasi-Newton methods** are more general optimization methods that approximate the Hessian in Newton's method.
- Various **convergence criteria** are available for terminating iterative optimization methods.

Motivation: Moving Targets (1/2)

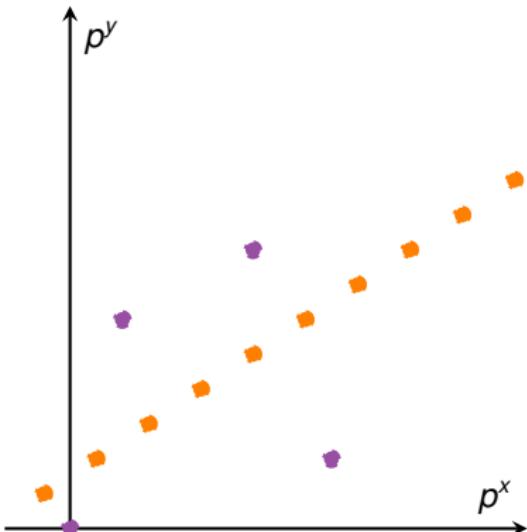


Motivation: Moving Targets (2/2)

- In practice, we often wish to track a moving target.
- One way is to recompute the position at every time step.
- This ignores the time continuity.
- We get a better result by modeling the temporal relationship of measurements.
- This can be done using (stochastic) differential equations and difference equations.

Localizing a Moving Target (1/4)

- Target moves rather than being stationary
- Sensors measure periodically, e.g., every second
- We can now either
 - ➊ recompute the position estimate at every time, or
 - ➋ use a dynamic model to connect the time points.



Localizing a Moving Target (2/4)

- Let us try a straight line model:

$$\begin{aligned} p^x(t) &= p^x(0) + v^x t, \\ p^y(t) &= p^y(0) + v^y t. \end{aligned}$$

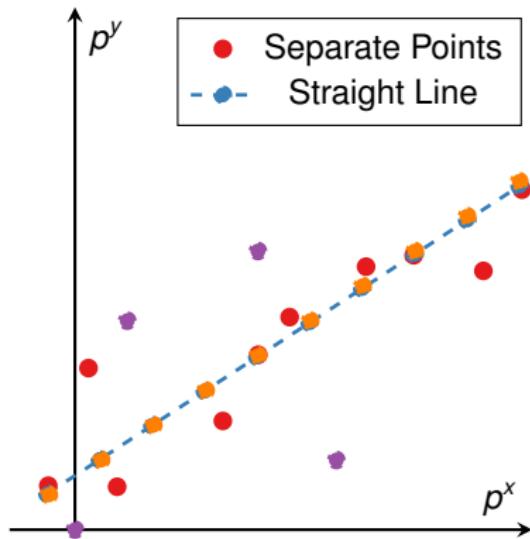
- Measurement model:

$$\begin{aligned} y_n(t) &= \sqrt{(p^x(t) - s_n^x)^2 + (p^y(t) - s_n^y)^2 + r_n(t)} \\ &= \sqrt{(p^x(0) + v^x t - s_n^x)^2 + (p^y(0) + v^y t - s_n^y)^2 + r_n(t)} \end{aligned}$$

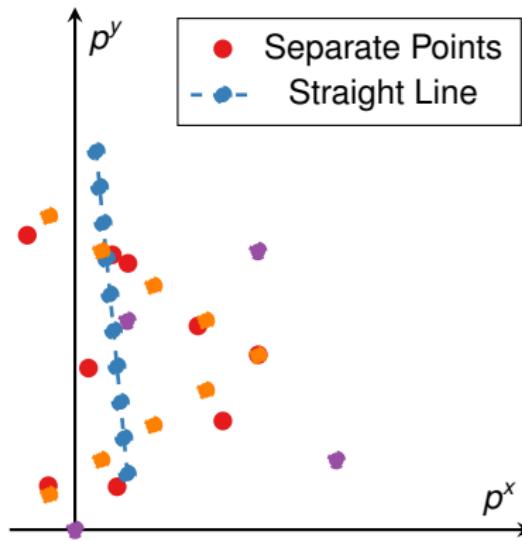
- We need to estimate 4 parameters:

$$\mathbf{x} = [p_t^x(0) \quad p_t^y(0) \quad v^x \quad v^y]^\top$$

Localizing a Moving Target (3/4)



Localizing a Moving Target (4/4)

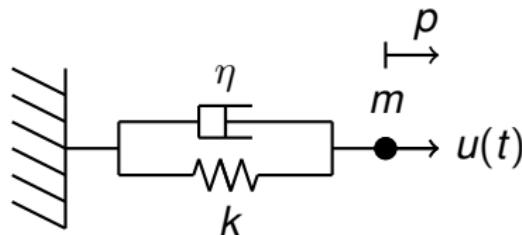


Localizing a Moving Target: Conclusions

- The static approach is not too well suited for time-varying processes
- A systematic method that relates (time-wise) related measurements is needed
- Solution: Use differential (and difference) equations to model the time-varying, i.e., dynamic, system

ODE Modeling of Dynamic Systems

- Ordinary differential equations (ODEs) can be used to describe many dynamic systems.
- Example: Spring-damper system:



- Second order ordinary differential equation:

$$ma(t) = -kp(t) - \eta v(t) + u(t)$$

- Other examples: Newtonian/Hamiltonian dynamics, kinematic models, heat and mass transfer, wave equations,

...

Example: State-Space Representation of ODEs

- The second order ODE for spring:

$$ma(t) = -kp(t) - \eta v(t) + u(t)$$

- Equation system representation:

$$\begin{bmatrix} v(t) \\ a(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{\eta}{m} \end{bmatrix} \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t)$$

- First order ODE equation system:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{\eta}{m} \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t)$$

- $\mathbf{x}(t) = [p(t) \quad v(t)]^T$ is the state of the system

Example: A Coffee Cup's Cooling (1/2)

- Newton's law of cooling for the coffee cup:

$$\frac{dT_c(t)}{dt} = -k_1(T_c(t) - T_r(t)),$$

- Newton's law of cooling for the room:

$$\frac{dT_r(t)}{dt} = -k_2(T_r(t) - T_a(t)) + h(t),$$

- Equation system:

$$\frac{dT_r(t)}{dt} = -k_2(T_r(t) - T_a(t)) + h(t)$$

$$\frac{dT_c(t)}{dt} = -k_1(T_c(t) - T_r(t))$$

Example: A Coffee Cup's Cooling (2/2)

- The equation system:

$$\frac{dT_r(t)}{dt} = -k_2(T_r(t) - T_a(t)) + h(t)$$

$$\frac{dT_c(t)}{dt} = -k_1(T_c(t) - T_r(t))$$

- In matrix form:

$$\begin{bmatrix} \frac{dT_r(t)}{dt} \\ \frac{dT_c(t)}{dt} \end{bmatrix} = \begin{bmatrix} -k_2 & 0 \\ k_1 & -k_1 \end{bmatrix} \begin{bmatrix} T_r(t) \\ T_c(t) \end{bmatrix} + \begin{bmatrix} k_2 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} T_a(t) \\ h(t) \end{bmatrix}$$

- Compact state-space notation:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u\mathbf{u}(t)$$

A Linear System of Differential Equations (1/2)

General system of first order differential equations:

$$\begin{aligned}\dot{x}_1(t) &= a_{11}x_1(t) + a_{12}x_2(t) + \cdots + a_{1d_x}x_{d_x}(t) \\ &\quad + b_{11}u_1(t) + b_{12}u_2(t) + \cdots + b_{1d_u}u_{d_u}(t)\end{aligned}$$

$$\begin{aligned}\dot{x}_2(t) &= a_{21}x_1(t) + a_{22}x_2(t) + \cdots + a_{2d_x}x_{d_x}(t) \\ &\quad + b_{21}u_1(t) + b_{22}u_2(t) + \cdots + b_{2d_u}u_{d_u}(t)\end{aligned}$$

⋮

$$\begin{aligned}\dot{x}_{d_x}(t) &= a_{d_x1}x_1(t) + a_{d_x2}x_2(t) + \cdots + a_{d_xd_x}x_{d_x}(t) \\ &\quad + b_{d_x1}u_1(t) + b_{d_x2}u_2(t) + \cdots + b_{d_xd_u}u_{d_u}(t)\end{aligned}$$

A Linear System of Differential Equations (2/2)

- In matrix form:

$$\begin{bmatrix} \dot{x}_1(t) \\ \vdots \\ \dot{x}_{d_x}(t) \end{bmatrix} = \begin{bmatrix} a_{11} & \dots & a_{1d_x} \\ \vdots & \ddots & \vdots \\ a_{d_x 1} & \dots & a_{d_x d_x} \end{bmatrix} \begin{bmatrix} x_1(t) \\ \vdots \\ x_{d_x}(t) \end{bmatrix} + \begin{bmatrix} b_{11} & \dots & b_{1d_u} \\ \vdots & \ddots & \vdots \\ b_{d_x 1} & \dots & b_{d_x d_u} \end{bmatrix} \begin{bmatrix} u_1(t) \\ \vdots \\ u_{d_u}(t) \end{bmatrix}$$

- Compact state-space notation:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u\mathbf{u}(t)$$

- This is called the state-space form of the differential equation system, $\mathbf{x}(t)$ is the state of the system

Transforming ODEs to State-Space Form (1/2)

- L th order ODE in $z(t)$

$$\frac{d^L z(t)}{dt^L} = c_0 z(t) + c_2 \frac{dz(t)}{dt} + \cdots + c_{L-1} \frac{d^{L-1} z(t)}{dt^{L-1}} + d_1 u(t)$$

- Choose state components:

$$x_1(t) = z(t), \quad x_2(t) = \frac{dz(t)}{dt}, \quad \dots, \quad x_{d_x}(t) = \frac{d^{L-1} z(t)}{dt^{L-1}}$$

- Then we have:

$$\dot{x}_1(t) = \frac{dz(t)}{dt} = x_2(t)$$

$$\dot{x}_2(t) = \frac{d^2 z(t)}{dt^2} = x_3(t)$$

$$\vdots$$

$$\dot{x}_{d_x}(t) = \frac{d^L z(t)}{dt^L} = c_0 z(t) + c_2 \frac{dz(t)}{dt} + \cdots + c_{L-1} \frac{d^{L-1} z(t)}{dt^{L-1}} + d_1 u(t)$$

Transforming ODEs to State-Space Form (2/2)

- Rewritten in terms of states x_i :

$$\dot{x}_1(t) = x_2(t)$$

 \vdots

$$\dot{x}_{d_x}(t) = c_0x_1(t) + c_1x_2(t) + \cdots + c_{L-1}x_{d_x}(t) + d_1u(t)$$

- In matrix form:

$$\underbrace{\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \vdots \\ \dot{x}_{d_x}(t) \end{bmatrix}}_{\triangleq \dot{\mathbf{x}}(t)} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & & \vdots \\ \vdots & & & \ddots & 0 \\ c_0 & c_1 & \dots & c_{L-1} & \end{bmatrix}}_{\triangleq \mathbf{A}} \underbrace{\begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_{d_x}(t) \end{bmatrix}}_{\triangleq \mathbf{x}(t)} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ d_1 \end{bmatrix}}_{\triangleq \mathbf{B}_u} u(t).$$

Deterministic Linear State-Space Model

- The dynamic model describes the evolution of the state:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

- The measurement model relates the state $\mathbf{x}_n = \mathbf{x}(t_n)$ at t_n to the measurement \mathbf{y}_n
- The linear measurement model is

$$\mathbf{y}_n = \mathbf{G}\mathbf{x}_n + \mathbf{r}_n.$$

- The deterministic linear state-space model combines the linear dynamic and measurement models

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u\mathbf{u}(t),$$

$$\mathbf{y}_n = \mathbf{G}\mathbf{x}_n + \mathbf{r}_n.$$

Example: A Car Navigating in 2D (1)

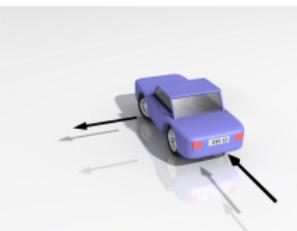
- Newton's law gives:

$$m a^x = F_p^x$$

$$m a^y = F_p^y$$

- Defining state $\mathbf{x} = [p^x \ p^y \ v^x \ v^y]^T$ leads to

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1/m & 0 \\ 0 & 1/m \end{bmatrix} \begin{bmatrix} F_p^x \\ F_p^y \end{bmatrix}$$



- Assuming position measurements \mathbf{y}_n gives

$$\mathbf{y}_n = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \mathbf{x}_n + \mathbf{r}_n.$$

Uncertainty in Dynamic Models

- The deterministic input $u(t)$ might **not be known**
- The model does not capture **every aspect** of the process
- Solution: Add a **stochastic process** $w(t)$ as an input
- Example: **Stochastic differential equation** (SDE) of order L :

$$\frac{d^L z(t)}{dt^L} = c_0 z(t) + c_1 \frac{dz(t)}{dt} + \dots + c_{L-1} \frac{d^{L-1} z(t)}{dt^{L-1}} + d_1 w(t)$$

Input Process $w(t)$

- Assumed to be zero-mean and stationary
- Characterized by its autocorrelation function...

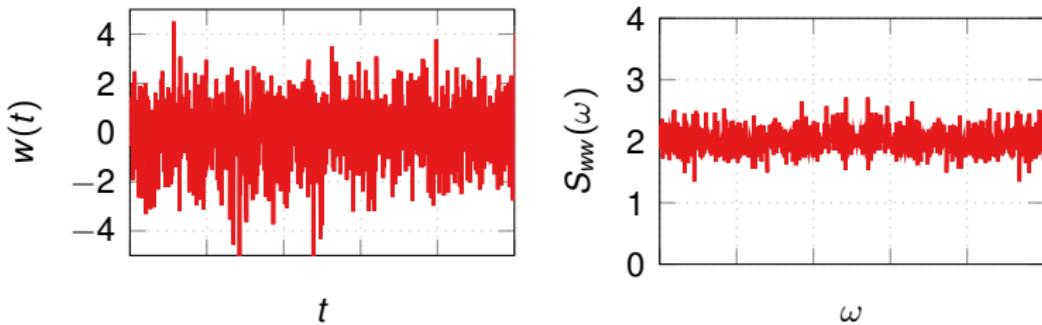
$$R_{ww}(\tau) = E\{w(t + \tau)w(t)\}$$

- ... or its power spectral density

$$S_{ww}(\omega) = \int R_{ww}(\tau) e^{-i\omega\tau} d\tau$$

White Processes

- “White noise” — equal contributions of each frequency
- Autocorrelation function: $R_{ww}(\tau) = \sigma_w^2 \delta(\tau)$
- Power spectral density: $S_{ww} = \sigma_w^2$
- Many forms of colored noise are filtered versions of white noise



Stochastic Linear State-Space Model

- Derivation of the stochastic dynamic model follows the same steps as for the deterministic case
- The stochastic process $\mathbf{w}(t)$ takes the place of the deterministic input $\mathbf{u}(t)$
- A system can have both deterministic and stochastic inputs
- Linear stochastic dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_w\mathbf{w}(t)$$

- Linear stochastic state-space model with measurements:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_w\mathbf{w}(t)$$

$$\mathbf{y}_n = \mathbf{G}\mathbf{x}_n + \mathbf{r}_n$$

Example: A Car Navigating in 2D (2)

- Recall the deterministic dynamic model:

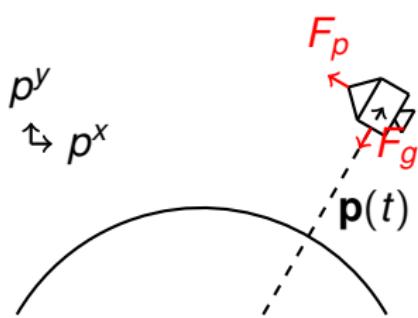
$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1/m & 0 \\ 0 & 1/m \end{bmatrix} \begin{bmatrix} F_p^x \\ F_p^y \end{bmatrix}$$

- F_p^x, F_p^y might be unknown when localizing the car
- Assume stochastic processes as the input:

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} w_1(t) \\ w_2(t) \end{bmatrix}$$

- This is the Wiener velocity model in 2D

Example: Dynamic Model for a Spacecraft (1/2)



- Gravitational acceleration:

$$g \approx g_0 \left(\frac{r_e}{|\mathbf{p}(t)|} \right)^2,$$

Example: Dynamic Model for a Spacecraft (2/2)

- Gravitational pull: $\mathbf{F}_g = -mg_0 r_e^2 \frac{\mathbf{p}(t)}{|\mathbf{p}(t)|^3}$
- Propulsion: $\mathbf{F}_p = F_p \frac{1}{|\mathbf{p}(t)|} \begin{bmatrix} -p^y(t) \\ p^x(t) \end{bmatrix}$
- Differential equation:

$$m\mathbf{a}(t) = -mg_0 r_e^2 \frac{\mathbf{p}(t)}{|\mathbf{p}(t)|^3} + \frac{1}{|\mathbf{p}(t)|} \begin{bmatrix} -p^y(t) \\ p^x(t) \end{bmatrix} u(t).$$

- State vector:

$$\mathbf{x}(t) = [p^x(t) \quad p^y(t) \quad v^x(t) \quad v^y(t)]^\top.$$

Can not be written as $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u \mathbf{u}(t)$.

Nonlinear Differential Equation Systems

- Nonlinear ordinary differential equation system (b_{ij} may depend on $x_n(t)$):

$$\dot{x}_1(t) = f_1(x_1(t), x_2(t), \dots, x_{d_x}(t)) + b_{11}u_1(t) + \dots b_{1d_u}u_{d_u}(t)$$

$$\dot{x}_2(t) = f_2(x_1(t), x_2(t), \dots, x_{d_x}(t)) + b_{21}u_1(t) + \dots b_{2d_u}u_{d_u}(t)$$

⋮

$$\dot{x}_{d_x}(t) = f_{d_x}(x_1(t), x_2(t), \dots, x_{d_x}(t)) + b_{d_x 1}u_1(t) + \dots b_{d_x d_u}u_{d_u}(t)$$

- State vector: $\mathbf{x}(t) = [x_1(t) \quad x_2(t) \quad \dots \quad x_{d_x}(t)]^\top$

- In vector form:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \vdots \\ \dot{x}_{d_x}(t) \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{x}(t)) \\ f_2(\mathbf{x}(t)) \\ \vdots \\ f_{d_x}(\mathbf{x}(t)) \end{bmatrix} + \begin{bmatrix} b_{11}(\mathbf{x}(t)) & \dots & b_{1d_u}(\mathbf{x}(t)) \\ b_{21}(\mathbf{x}(t)) & & \vdots \\ \vdots & & \ddots \\ b_{d_x 1}(\mathbf{x}(t)) & \dots & b_{d_x d_u}(\mathbf{x}(t)) \end{bmatrix} \mathbf{u}(t).$$

Nonlinear Continuous-Time State-Space Models

- Deterministic nonlinear dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_u(\mathbf{x}(t))\mathbf{u}(t)$$

- Stochastic nonlinear dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

- Nonlinear measurement model:

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n$$

- Stochastic nonlinear state-space model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n$$

Example: Dynamic Model for a Spacecraft (2)

- Differential equation:

$$m\mathbf{a}(t) = -mg_0 r_e^2 \frac{\mathbf{p}(t)}{|\mathbf{p}(t)|^3} + \frac{1}{|\mathbf{p}(t)|} \begin{bmatrix} -p^y(t) \\ p^x(t) \end{bmatrix} w(t).$$

- State vector:

$$\mathbf{x}(t) = [p^x(t) \quad p^y(t) \quad v^x(t) \quad v^y(t)]^\top.$$

- Vector form:

$$\begin{aligned} \begin{bmatrix} v^x(t) \\ v^y(t) \\ a^x(t) \\ a^y(t) \end{bmatrix} &= \begin{bmatrix} v^x(t) \\ v^y(t) \\ -g_0 r_e^2 \frac{p^x(t)}{|\mathbf{p}(t)|^3} \\ -g_0 r_e^2 \frac{p^y(t)}{|\mathbf{p}(t)|^3} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{p^y(t)}{m|\mathbf{p}(t)|} \\ \frac{p^x(t)}{m|\mathbf{p}(t)|} \end{bmatrix} w(t) \\ &= \begin{bmatrix} f_1(\mathbf{x}(t)) \\ f_2(\mathbf{x}(t)) \\ f_3(\mathbf{x}(t)) \\ f_4(\mathbf{x}(t)) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{p^y(t)}{m|\mathbf{p}(t)|} \\ \frac{p^x(t)}{m|\mathbf{p}(t)|} \end{bmatrix} w(t), \end{aligned}$$

Example: Robot Navigation in 2D (1/3)

- Quasi-constant turn model:

$$\dot{p}^x(t) = v(t) \cos(\varphi(t))$$

$$\dot{p}^y(t) = v(t) \sin(\varphi(t))$$

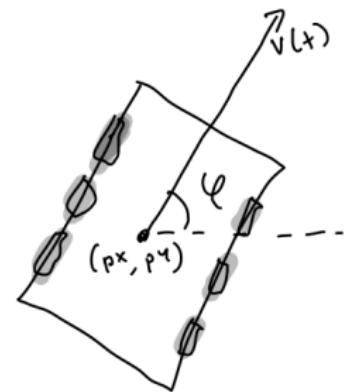
$$\dot{v}(t) = w_1(t)$$

$$\dot{\varphi}(t) = w_2(t)$$

- The state is

$$\mathbf{x}(t) = [p^x(t) \quad p^y(t) \quad v(t) \quad \varphi(t)]^\top.$$

- Position measurement: picks $p^x(t)$ and $p^y(t)$
- Speed measurements (odometry): $v(t)$
- Magnetometer (compass): $\varphi(t)$.



Example: Robot Navigation in 2D (2/3)

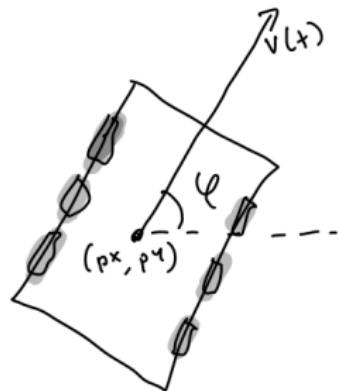
- Gyroscope measures $\dot{\varphi}(t)$.
- Accelerometer measures $\dot{\varphi}(t)$.
 - Word of warning: accelerometers are usually not accurate enough for this.
- Putting these into the equations we get the model

$$\dot{p}^x(t) = v(t) \cos(\varphi(t))$$

$$\dot{p}^y(t) = v(t) \sin(\varphi(t))$$

$$\dot{v}(t) = a_{\text{acc}}(t) + w_1(t)$$

$$\dot{\varphi}(t) = \omega_{\text{gyro}}(t) + w_2(t).$$



- The state is still

$$\mathbf{x}(t) = [p^x(t) \quad p^y(t) \quad v(t) \quad \varphi(t)]^\top.$$

Example: Robot Navigation in 2D (3/3)

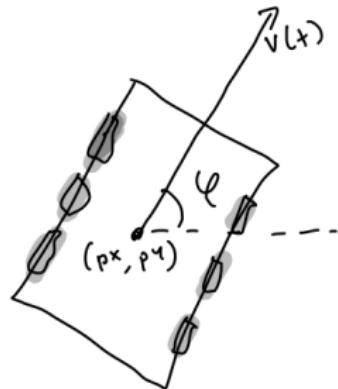
- Often we have the speed $v(t)$ directly available (e.g., from wheels)
- Then we can reduce the model to

$$\dot{p}^x(t) = v(t) \cos(\varphi(t))$$

$$\dot{p}^y(t) = v(t) \sin(\varphi(t))$$

$$\dot{\varphi}(t) = \omega_{\text{gyro}}(t) + w(t).$$

- The state is now
 $\mathbf{x}(t) = [p^x(t) \ p^y(t) \ \varphi(t)]^\top.$
- This is a typical model used in 2D tracking.



Summary

- Higher order ODEs and SDEs can be transformed to a first-order vector-valued equation system
- The deterministic linear state-space model is

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u\mathbf{u}(t)$$

$$\mathbf{y}_n = \mathbf{G}\mathbf{x}_n + \mathbf{r}_n$$

- The stochastic linear state-space model with stochastic input process $\mathbf{w}(t)$ is

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_w\mathbf{w}(t)$$

$$\mathbf{y}_n = \mathbf{G}\mathbf{x}_n + \mathbf{r}_n$$

- Nonlinear continuous-time state-space model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n$$



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Nonlinear Continuous-Time Models and Discrete-Time Dynamic Models

Simo Särkkä

Aalto University

October 30, 2020

Contents

- 1 Intended Learning Outcomes and Recap
- 2 Nonlinear State-Space Models
- 3 Linear Discrete-Time Models
- 4 Stochastic Linear Discrete-Time Models
- 5 Nonlinear Discrete-Time Models
- 6 Summary

Intended Learning Outcomes

After this lecture, you will be able to:

- construct nonlinear continuous-time state-space models,
- distinguish continuous-time and discrete-time models,
- construct discrete-time linear and non-linear state-space models.

Recap

- Higher order ODEs and SDEs can be transformed to a first-order vector-valued equation system
- The deterministic linear state-space model is

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u\mathbf{u}(t)$$

$$\mathbf{y}_n = \mathbf{G}\mathbf{x}_n + \mathbf{r}_n$$

- The stochastic linear state-space model with stochastic input process $\mathbf{w}(t)$ is

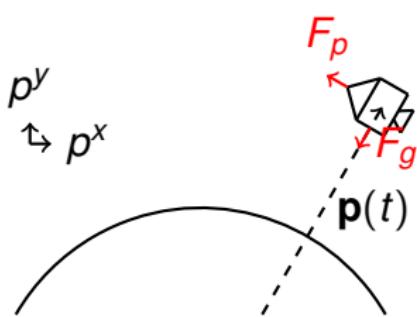
$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_w\mathbf{w}(t)$$

$$\mathbf{y}_n = \mathbf{G}\mathbf{x}_n + \mathbf{r}_n$$

- The 2D Wiener velocity model is

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} w_1(t) \\ w_2(t) \end{bmatrix}$$

Example: Dynamic Model for a Spacecraft (1/2)



- Gravitational acceleration:

$$g \approx g_0 \left(\frac{r_e}{|\mathbf{p}(t)|} \right)^2,$$

Example: Dynamic Model for a Spacecraft (2/2)

- Gravitational pull: $\mathbf{F}_g = -mg_0 r_e^2 \frac{\mathbf{p}(t)}{|\mathbf{p}(t)|^3}$
- Propulsion: $\mathbf{F}_p = F_p \frac{1}{|\mathbf{p}(t)|} \begin{bmatrix} -p^y(t) \\ p^x(t) \end{bmatrix}$
- Differential equation:

$$m\mathbf{a}(t) = -mg_0 r_e^2 \frac{\mathbf{p}(t)}{|\mathbf{p}(t)|^3} + \frac{1}{|\mathbf{p}(t)|} \begin{bmatrix} -p^y(t) \\ p^x(t) \end{bmatrix} u(t).$$

- State vector:

$$\mathbf{x}(t) = [p^x(t) \quad p^y(t) \quad v^x(t) \quad v^y(t)]^\top.$$

Can not be written as $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u \mathbf{u}(t)$.

Nonlinear Differential Equation Systems

- Nonlinear ordinary differential equation system (b_{ij} may depend on $x_n(t)$):

$$\dot{x}_1(t) = f_1(x_1(t), x_2(t), \dots, x_{d_x}(t)) + b_{11}u_1(t) + \dots b_{1d_u}u_{d_u}(t)$$

$$\dot{x}_2(t) = f_2(x_1(t), x_2(t), \dots, x_{d_x}(t)) + b_{21}u_1(t) + \dots b_{2d_u}u_{d_u}(t)$$

⋮

$$\dot{x}_{d_x}(t) = f_{d_x}(x_1(t), x_2(t), \dots, x_{d_x}(t)) + b_{d_x 1}u_1(t) + \dots b_{d_x d_u}u_{d_u}(t)$$

- State vector: $\mathbf{x}(t) = [x_1(t) \quad x_2(t) \quad \dots \quad x_{d_x}(t)]^T$

- In vector form:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \vdots \\ \dot{x}_{d_x}(t) \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{x}(t)) \\ f_2(\mathbf{x}(t)) \\ \vdots \\ f_{d_x}(\mathbf{x}(t)) \end{bmatrix} + \begin{bmatrix} b_{11}(\mathbf{x}(t)) & \dots & b_{1d_u}(\mathbf{x}(t)) \\ b_{21}(\mathbf{x}(t)) & & \vdots \\ \vdots & \ddots & \vdots \\ b_{d_x 1}(\mathbf{x}(t)) & \dots & b_{d_x d_u}(\mathbf{x}(t)) \end{bmatrix} \mathbf{u}(t).$$

Nonlinear Continuous-Time State-Space Models

- Deterministic nonlinear dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_u(\mathbf{x}(t))\mathbf{u}(t)$$

- Stochastic nonlinear dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

- Nonlinear measurement model:

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n$$

- Stochastic nonlinear state-space model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n$$

Example: Dynamic Model for a Spacecraft (2)

- Differential equation:

$$m\mathbf{a}(t) = -mg_0 r_e^2 \frac{\mathbf{p}(t)}{|\mathbf{p}(t)|^3} + \frac{1}{|\mathbf{p}(t)|} \begin{bmatrix} -p^y(t) \\ p^x(t) \end{bmatrix} w(t).$$

- State vector:

$$\mathbf{x}(t) = [p^x(t) \quad p^y(t) \quad v^x(t) \quad v^y(t)]^\top.$$

- Vector form:

$$\begin{aligned} \begin{bmatrix} v^x(t) \\ v^y(t) \\ a^x(t) \\ a^y(t) \end{bmatrix} &= \begin{bmatrix} v^x(t) \\ v^y(t) \\ -g_0 r_e^2 \frac{p^x(t)}{|\mathbf{p}(t)|^3} \\ -g_0 r_e^2 \frac{p^y(t)}{|\mathbf{p}(t)|^3} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{p^y(t)}{m|\mathbf{p}(t)|} \\ \frac{p^x(t)}{m|\mathbf{p}(t)|} \end{bmatrix} w(t) \\ &= \begin{bmatrix} f_1(\mathbf{x}(t)) \\ f_2(\mathbf{x}(t)) \\ f_3(\mathbf{x}(t)) \\ f_4(\mathbf{x}(t)) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{p^y(t)}{m|\mathbf{p}(t)|} \\ \frac{p^x(t)}{m|\mathbf{p}(t)|} \end{bmatrix} w(t), \end{aligned}$$

Example: Robot Navigation in 2D (1/4)

- Quasi-constant turn model:

$$\dot{p}^x(t) = v(t) \cos(\varphi(t))$$

$$\dot{p}^y(t) = v(t) \sin(\varphi(t))$$

$$\dot{v}(t) = w_1(t)$$

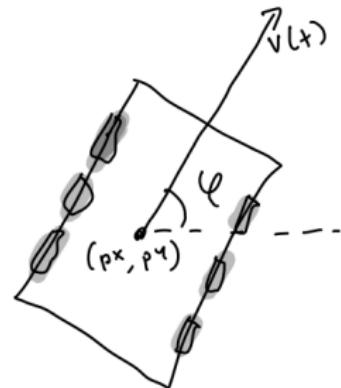
$$\dot{\varphi}(t) = w_2(t)$$



- The state is

$$\mathbf{x}(t) = [p^x(t) \quad p^y(t) \quad v(t) \quad \varphi(t)]^\top.$$

- Position measurement: picks $p^x(t)$ and $p^y(t)$
- Speed measurements (odometry): $v(t)$
- Magnetometer (compass): $\varphi(t)$.



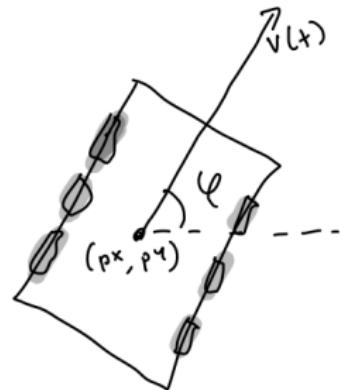
Example: Robot Navigation in 2D (2/4)

- Gyroscope measures $\dot{\varphi}(t)$.
- Accelerometer measures $\dot{\varphi}(t)$.
 - Word of warning: accelerometers are usually not accurate enough for this.
- Putting these into the equations we get the model



$$\begin{aligned}\dot{p}^x(t) &= v(t) \cos(\varphi(t)) \\ \dot{p}^y(t) &= v(t) \sin(\varphi(t)) \\ \dot{v}(t) &= a_{\text{acc}}(t) + w_1(t) \\ \dot{\varphi}(t) &= \omega_{\text{gyro}}(t) + w_2(t).\end{aligned}$$

- The state is still
- $$\mathbf{x}(t) = [p^x(t) \quad p^y(t) \quad v(t) \quad \varphi(t)]^\top.$$



Example: Robot Navigation in 2D (3/4)

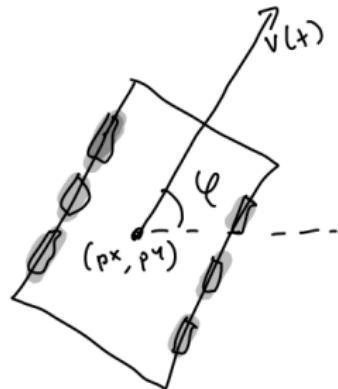
- Often we have the speed $v(t)$ directly available (e.g., from wheels)
- Then we can reduce the model to

$$\dot{p}^x(t) = v(t) \cos(\varphi(t))$$

$$\dot{p}^y(t) = v(t) \sin(\varphi(t))$$

$$\dot{\varphi}(t) = \omega_{\text{gyro}}(t) + w(t).$$

- The state is now
 $\mathbf{x}(t) = [p^x(t) \quad p^y(t) \quad \varphi(t)]^\top.$
- This is a typical model used in 2D tracking.



Example: Robot Navigation in 2D (4/4)

- Finally, the speed measurement is often not accurate.
- Thus it is beneficial to include additional noises to the dynamic model:

$$\dot{p}^x(t) = v(t) \cos(\varphi(t)) + w_1(t)$$

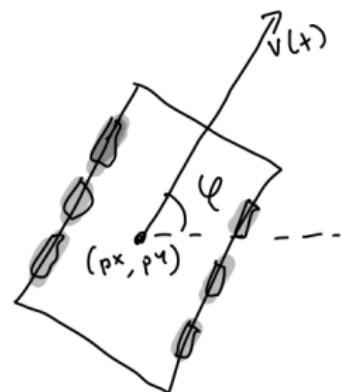
$$\dot{p}^y(t) = v(t) \sin(\varphi(t)) + w_2(t)$$

$$\dot{\varphi}(t) = \omega_{\text{gyro}}(t) + w_3(t).$$

- The state is still

$$\mathbf{x}(t) = [p^x(t) \quad p^y(t) \quad \varphi(t)]^T.$$

- This model would be a good candidate for the dynamic model in the project work.



Discrete-Time Processes and Difference Equations

- Some processes are only defined at discrete time points t_1, t_2, \dots
- The discrete-time equivalent of differential equations are difference equations
- The difference of two discrete points in time takes the role of the derivative

Vector Form of Difference Equation Systems

- Equation system of d_x linear difference equations:

$$x_{1,n} = a_{11}x_{1,n-1} + \cdots + a_{1d_x}x_{d_x,n-1} + b_{11}u_{1,n} + \cdots + b_{1d_u}u_{d_u,n}$$

$$x_{2,n} = a_{21}x_{1,n-1} + \cdots + a_{2d_x}x_{d_x,n-1} + b_{21}u_{1,n} + \cdots + b_{2d_u}u_{d_u,n}$$

⋮

$$x_{d_x,n} = a_{d_x1}x_{1,n-1} + \cdots + a_{d_xd_x}x_{d_x,n-1} + b_{d_x1}u_{1,n} + \cdots + b_{d_xd_u}u_{d_u,n}$$

- Vector form:

$$\begin{bmatrix} x_{1,n} \\ \vdots \\ x_{d_x,n} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1d_x} \\ \vdots & \ddots & \vdots \\ a_{d_x1} & \cdots & a_{d_xd_x} \end{bmatrix} \begin{bmatrix} x_{1,n-1} \\ \vdots \\ x_{d_x,n-1} \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{1d_u} \\ \vdots & \ddots & \vdots \\ b_{d_x1} & \cdots & b_{d_xd_u} \end{bmatrix} \begin{bmatrix} u_{1,n} \\ \vdots \\ u_{d_u,n} \end{bmatrix}$$

- Compact notation:

$$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{B}_u\mathbf{u}_n$$

Deterministic Discrete-Time State-Space Model

- Linear discrete-time dynamic model:

$$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{B}_u \mathbf{u}_n$$

- Deterministic, linear discrete-time state-space model:

$$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{B}_u \mathbf{u}_n$$

$$\mathbf{y}_n = \mathbf{G}\mathbf{x}_n + \mathbf{r}_n.$$

with $E\{\mathbf{r}_n\} = 0$, $\text{Cov}\{\mathbf{r}_n\} = \mathbf{R}_n$, $\text{Cov}\{\mathbf{r}_n, \mathbf{r}_m\} = 0$ ($n \neq m$)

Conversion of L th Order Difference Equation (1/2)

- L th order difference equation (with single input u_n):

$$Z_n = c_1 Z_{n-1} + c_2 Z_{n-2} + \cdots + c_L Z_{n-L} + d_1 u_n$$

- It is easier to choose \mathbf{x}_{n-1} on the RHS
(c.f. continuous case)
- A possible choice:

$$x_{1,n-1} = Z_{n-1}, x_{2,n-1} = Z_{n-2}, \dots, x_{d_x,n-1} = Z_{n-L}.$$

Conversion of L th Order Difference Equation (2/2)

- Difference equation system:

$$x_{1,n} = c_1 x_{1,n-1} + c_2 x_{2,n-1} + \cdots + c_L x_{d_x,n-1} + d_1 u_n$$

$$x_{2,n} = z_{n-1} = x_{1,n-1}$$

⋮

$$x_{d_x,n} = z_{n-L+1} = x_{d_x+1,n-1}$$

- Vector form:

$$\begin{bmatrix} x_{1,n} \\ x_{2,n} \\ \vdots \\ x_{d_x,n} \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & \dots & c_L \\ 1 & 0 & & \vdots \\ \vdots & \ddots & & \\ 0 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{1,n-1} \\ x_{2,n-1} \\ \vdots \\ x_{d_x,n-1} \end{bmatrix} + \begin{bmatrix} d_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} u_n,$$

Stochastic Linear State-Space Model (1/2)

- Dynamics are not entirely deterministic and inputs may not always be known
- Let the **process noise** \mathbf{q}_n (random variable) take the place of the input \mathbf{u}_n (or in addition to \mathbf{u}_n)
- Stochastic linear discrete-time dynamic model:

$$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{B}_q\mathbf{q}_n$$

Stochastic Linear State-Space Model (2/2)

- Stochastic linear discrete-time dynamic model:

$$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{B}_q \mathbf{q}_n$$

- The process noise follows

$$\mathbf{q}_n \sim p(\mathbf{q}_n)$$

with $E\{\mathbf{q}_n\} = 0$, $\text{Cov}\{\mathbf{q}_n\} = \mathbf{Q}_n$, and $\text{Cov}\{\mathbf{q}_m, \mathbf{q}_n\} = 0$ ($m \neq n$)

- Stochastic linear discrete-time state-space model:

$$\mathbf{x}_n = \mathbf{F}\mathbf{x}_{n-1} + \mathbf{B}_q \mathbf{q}_n$$

$$\mathbf{y}_n = \mathbf{G}\mathbf{x}_n + \mathbf{r}_n$$

Nonlinear Discrete-Time Dynamic Model

- Difference equations may also be nonlinear
- Nonlinear difference equation system (with process noise inputs):

$$x_{1,n} = f_1(x_{1,n-1}, x_{2,n-1}, \dots, x_{d_x,n-1}) + b_{11}q_{1,n} + \dots + b_{1d_q}q_{d_q,n}$$

$$x_{2,n} = f_2(x_{1,n-1}, x_{2,n-1}, \dots, x_{d_x,n-1}) + b_{21}q_{1,n} + \dots + b_{2d_q}q_{d_q,n}$$

$$\vdots$$

$$x_{d_x,n} = f_{d_x}(x_{1,n-1}, x_{2,n-1}, \dots, x_{d_x,n-1}) + b_{d_x1}q_{1,n} + \dots + b_{d_xd_q}q_{d_q,n}$$

- Vector form:

$$\begin{bmatrix} x_{1,n} \\ \vdots \\ x_{d_x,n} \end{bmatrix} = \begin{bmatrix} f_1(x_{1,n-1}, x_{2,n-1}, \dots, x_{d_x,n-1}) \\ \vdots \\ f_{d_x}(x_{1,n-1}, x_{2,n-1}, \dots, x_{d_x,n-1}) \end{bmatrix} + \begin{bmatrix} b_{11} & \dots & b_{1d_u} \\ \vdots & \ddots & \vdots \\ b_{d_x1} & \dots & b_{d_xd_u} \end{bmatrix} \begin{bmatrix} q_{1,n} \\ \vdots \\ q_{d_u,n} \end{bmatrix}$$

Nonlinear Discrete-Time State-Space Model

- Compact notation of the dynamic model:

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{B}_q \mathbf{q}_n$$

- Nonlinear discrete-time state-space model:

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{B}_q \mathbf{q}_n$$

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n$$

where:

- $\mathbf{q}_n \sim p(\mathbf{q}_n)$, $E\{\mathbf{q}_n\} = 0$, $Cov\{\mathbf{q}_n\} = \mathbf{Q}_n$
- $\mathbf{r}_n \sim p(\mathbf{r}_n)$, $E\{\mathbf{r}_n\} = 0$, $Cov\{\mathbf{r}_n\} = \mathbf{R}_n$

Summary

- Nonlinear continuous-time state-space model:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t) \\ \mathbf{y}_n &= \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n\end{aligned}$$

- Linear discrete-time state-space model:

$$\begin{aligned}\mathbf{x}_n &= \mathbf{F}\mathbf{x}_{n-1} + \mathbf{B}_q\mathbf{q}_n \\ \mathbf{y}_n &= \mathbf{G}\mathbf{x}_n + \mathbf{r}_n\end{aligned}$$

- Nonlinear discrete-time state-space model:

$$\begin{aligned}\mathbf{x}_n &= \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{B}_q(\mathbf{x}_{n-1})\mathbf{q}_n \\ \mathbf{y}_n &= \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n\end{aligned}$$



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Discretization of Continuous-Time Dynamic Models

Simo Särkkä

Aalto University

November 6, 2020

Contents

- 1 Intended Learning Outcomes and Recap
- 2 Discretization of Linear ODEs
- 3 Discretization of Linear SDEs
- 4 Discretization of Non-Linear Systems
- 5 Summary

Intended Learning Outcomes

After this lecture, you will be able to:

- explain why continuous-time dynamic models need to be discretized in practice
- construct discrete-time dynamic models from linear ODE and SDE state-space models
- construct approximate discrete-time dynamic models from non-linear ODE and SDE models

Recap

- Nonlinear continuous-time state-space model:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t) \\ \mathbf{y}_n &= \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n\end{aligned}$$

- Linear discrete-time state-space model:

$$\begin{aligned}\mathbf{x}_n &= \mathbf{F}\mathbf{x}_{n-1} + \mathbf{B}_q\mathbf{q}_n \\ \mathbf{y}_n &= \mathbf{G}\mathbf{x}_n + \mathbf{r}_n\end{aligned}$$

- Nonlinear discrete-time state-space model:

$$\begin{aligned}\mathbf{x}_n &= \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{B}_q(\mathbf{x}_{n-1})\mathbf{q}_n \\ \mathbf{y}_n &= \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n\end{aligned}$$

Discretization of Continuous-Time Models: Why?

- Sensor fusion is implemented in digital computers
- Data is only processed at t_1, t_2, \dots, t_n
- Discretized continuous-time models are closely related to discrete-time models
- Example: Vehicle tracking

Discretization of continuous-time models is equivalent to solving the ODE/SDE model between t_{n-1} and t_n

Solving Linear First Order ODEs (1/2)

- Goal: Solve the first order ODE

$$\dot{x}(t) = ax(t) + bu(t),$$

on the interval $(t_{n-1}, t_n]$.

- Ansatz: Multiply by the integrating factor e^{-at}

$$e^{-at}\dot{x}(t) = e^{-at}ax(t) + e^{-at}bu(t)$$

i.e.

$$e^{-at}\dot{x}(t) - e^{-at}ax(t) = e^{-at}bu(t)$$

- We can then identify the derivative on the left hand side:

$$\frac{d}{dt} [e^{-at}x(t)] = e^{-at}\dot{x}(t) - e^{-at}ax(t).$$

- Thus we have

$$\frac{d}{dt} [e^{-at}x(t)] = e^{-at}bu(t).$$

Solving Linear First Order ODEs (2/2)

- We can now integrate the both sides:

$$\int_{t_{n-1}}^{t_n} \frac{d}{dt} [e^{-at} x(t)] dt = \int_{t_{n-1}}^{t_n} e^{-at} bu(t) dt$$

- Solution:

$$e^{-at_n} x(t_n) - e^{-at_{n-1}} x(t_{n-1}) = \int_{t_{n-1}}^{t_n} e^{-at} bu(t) dt$$

- Rearranged:

$$x(t_n) = e^{a(t_n - t_{n-1})} x(t_{n-1}) + \int_{t_{n-1}}^{t_n} e^{a(t_n - t)} bu(t) dt$$

- Defining $\Delta t = t_n - t_{n-1}$ this is

$$x(t_n) = e^{a\Delta t} x(t_{n-1}) + \int_{t_{n-1}}^{t_{n-1} + \Delta t} e^{a(t_{n-1} + \Delta t - t)} bu(t) dt$$

Vector-Valued Linear First Order ODE

- General linear dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u\mathbf{u}(t)$$

- This is a vector-valued first order ODE

What is the integrating factor for vector-valued first order ODEs?

Matrix Exponential

- Definition of the exponential:

$$e^a = \sum_{k=0}^{\infty} \frac{1}{k!} a^k$$

- Definition of the matrix exponential:

$$e^{\mathbf{A}} = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{A}^k$$

- Derivative of matrix exponential w.r.t. scalar t :

$$\frac{d}{dt} e^{\mathbf{A}t} = e^{\mathbf{A}t} \mathbf{A}$$

- Matrix exponential of \mathbf{A}^T :

$$(e^{\mathbf{A}})^T = e^{\mathbf{A}^T}$$

Solving Linear First Order Vector ODEs (1/2)

- General linear dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u\mathbf{u}(t)$$

- Multiplication by the integrating factor $e^{-\mathbf{A}t}$:

$$e^{-\mathbf{A}t}\dot{\mathbf{x}}(t) = e^{-\mathbf{A}t}\mathbf{A}\mathbf{x}(t) + e^{-\mathbf{A}t}\mathbf{B}_u\mathbf{u}(t)$$

- Rearranging:

$$e^{-\mathbf{A}t}\dot{\mathbf{x}}(t) - e^{-\mathbf{A}t}\mathbf{A}\mathbf{x}(t) = e^{-\mathbf{A}t}\mathbf{B}_u\mathbf{u}(t)$$

- Substituting $\frac{d}{dt}e^{-\mathbf{A}t}\mathbf{x}(t) = e^{-\mathbf{A}t}\dot{\mathbf{x}}(t) - e^{-\mathbf{A}t}\mathbf{A}\mathbf{x}(t)$:

$$\frac{d}{dt}e^{-\mathbf{A}t}\mathbf{x}(t) = e^{-\mathbf{A}t}\mathbf{B}_u\mathbf{u}(t)$$

Solving Linear First Order Vector ODEs (2/2)

- We now have the ODE:

$$\frac{d}{dt} e^{-\mathbf{A}t} \mathbf{x}(t) = e^{-\mathbf{A}t} \mathbf{B}_u \mathbf{u}(t)$$

- Integration w.r.t. t :

$$\int_{t_{n-1}}^{t_n} d \left[e^{-\mathbf{A}t} \mathbf{x}(t) \right] = \int_{t_{n-1}}^{t_n} e^{-\mathbf{A}t} \mathbf{B}_u \mathbf{u}(t) dt$$

$$\left[e^{-\mathbf{A}t} \mathbf{x}(t) \right]_{t=t_{n-1}}^{t_n} = \int_{t_{n-1}}^{t_n} e^{-\mathbf{A}t} \mathbf{B}_u \mathbf{u}(t) dt$$

$$e^{-\mathbf{A}t_n} \mathbf{x}(t_n) - e^{-\mathbf{A}t_{n-1}} \mathbf{x}(t_{n-1}) = \int_{t_{n-1}}^{t_n} e^{-\mathbf{A}t} \mathbf{B}_u \mathbf{u}(t) dt$$

- Rearranging:

$$\mathbf{x}(t_n) = e^{\mathbf{A}(t_n-t_{n-1})} \mathbf{x}(t_{n-1}) + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_u \mathbf{u}(t) dt$$

Zero-Order-Hold Inputs

- Solution:

$$\mathbf{x}(t_n) = e^{\mathbf{A}(t_n - t_{n-1})} \mathbf{x}(t_{n-1}) + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_u \mathbf{u}(t) dt,$$

- The input $\mathbf{u}(t)$ can be often assumed to be constant between sampling instants (zero-order-hold; ZOH)
- Then:

$$\int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_u \mathbf{u}(t) dt = \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_u dt \mathbf{u}(t_{n-1})$$

Discretized Deterministic Linear Dynamic Model

- Linear continuous-time dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u\mathbf{u}(t)$$

- Discretized dynamic model:

$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{L}_n \mathbf{u}_{n-1},$$

where

$$\mathbf{F}_n \triangleq e^{\mathbf{A}(t_n - t_{n-1})}$$

$$\mathbf{L}_n \triangleq \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_u dt$$

The discretized dynamic model is completely equivalent to the continuous-time model

Example: Deterministic 1D Motion Model (1/4)

- Dynamic model:

$$\begin{bmatrix} \dot{p}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

- Recall:

$$\mathbf{F}_n = e^{\mathbf{A}\Delta t} = \sum_{j=0}^{\infty} \frac{1}{j!} \mathbf{A}^j (\Delta t)^j$$

Example: Deterministic 1D Motion Model (2/4)

- Powers of \mathbf{A} :

$$\mathbf{A}^0 = \mathbf{I}$$

$$\mathbf{A}^1 = \mathbf{A}$$

$$\mathbf{A}^j = \mathbf{0} \quad j \geq 2$$

- Hence:

$$\begin{aligned}\mathbf{F}_n &= \sum_{j=0}^{\infty} \frac{1}{j!} \mathbf{A}^j (\Delta t)^j = \frac{1}{0!} \mathbf{I} (\Delta t)^0 + \frac{1}{1!} \mathbf{A} \Delta t \\ &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}\end{aligned}$$

Example: Deterministic 1D Motion Model (3/4)

- Input matrix:

$$\mathbf{L}_n = \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_u dt$$

where:

$$e^{\mathbf{A}(t_n - t)} = \begin{bmatrix} 1 & t_n - t \\ 0 & 1 \end{bmatrix}, \quad \mathbf{B}_u = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Example: Deterministic 1D Motion Model (4/4)

- Continuous-time model:

$$\begin{bmatrix} \dot{p}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

- Discretized model:

$$\mathbf{x}_n = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \mathbf{x}_{n-1} + \begin{bmatrix} \frac{(\Delta t)^2}{2} \\ \Delta t \end{bmatrix} \mathbf{u}_{n-1}$$

Stochastic Linear Dynamic Model

- Stochastic linear dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_w\mathbf{w}(t),$$

- The only difference to the deterministic model is the input $\mathbf{u}(t)$ ($\mathbf{w}(t)$)
- $\mathbf{w}(t)$ is a zero-mean white stochastic process
- Auto-correlation function:

$$R_{ww}(\tau) = E\{\mathbf{w}(t + \tau)\mathbf{w}(t)\} = \boldsymbol{\Sigma}_w \delta(\tau)$$

- Here $\boldsymbol{\Sigma}_w$ is the spectral density of the white noise.
- Hence:

$$\mathbf{x}(t_n) = e^{\mathbf{A}(t_n - t_{n-1})} \mathbf{x}(t_{n-1}) + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_w \mathbf{w}(t) dt$$

Integration of the Stochastic Process

- Model:

$$\mathbf{x}(t_n) = e^{\mathbf{A}(t_n - t_{n-1})} \mathbf{x}(t_{n-1}) + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_w \mathbf{w}(t) dt$$

- $\mathbf{w}(t)$ is stochastic; not ZOH and not even integrable (with standard tools)
- Define a random variable as the process noise:

$$\mathbf{q}_n \triangleq \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_w \mathbf{w}(t) dt$$

- Then:

$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{q}_n$$

Mean of the Process Noise

- Process noise:

$$\mathbf{q}_n \triangleq \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_w \mathbf{w}(t) dt.$$

- We get

$$\begin{aligned}\mathbb{E}\{\mathbf{q}_n\} &= \mathbb{E}\left\{ \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_w \mathbf{w}(t) dt \right\} \\ &= \int_{t_{n-1}}^{t_n} \mathbb{E}\left\{ e^{\mathbf{A}(t_n-t)} \mathbf{B}_w \mathbf{w}(t) \right\} dt \\ &= \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_w \mathbb{E}\{\mathbf{w}(t)\} dt \\ &= 0.\end{aligned}$$

Covariance of the Process Noise (1/2)

- Process noise:

$$\mathbf{q}_n \triangleq \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_w \mathbf{w}(t) dt$$

- Covariance:

$$\text{Cov}\{\mathbf{q}_n\}$$

$$= E\{(\mathbf{q}_n - E\{\mathbf{q}_n\})(\mathbf{q}_n - E\{\mathbf{q}_n\})^\top\}$$

$$= E\{\mathbf{q}_n \mathbf{q}_n^\top\}$$

$$= E \left\{ \left(\int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_w \mathbf{w}(t) dt \right) \left(\int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-\tau)} \mathbf{B}_w \mathbf{w}(\tau) d\tau \right)^\top \right\}$$

$$= \int_{t_{n-1}}^{t_n} \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_w E\{\mathbf{w}(t)\mathbf{w}(\tau)^\top\} \mathbf{B}_w^\top (e^{\mathbf{A}(t_n-\tau)})^\top d\tau dt \dots$$

Covariance of the Process Noise (2/2)

- Process noise:

$$\mathbf{q}_n \triangleq \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_w \mathbf{w}(t) dt$$

- Covariance:

$$\begin{aligned}\text{Cov}\{\mathbf{q}_n\} &= \int_{t_{n-1}}^{t_n} \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_w E \left\{ \mathbf{w}(t) \mathbf{w}(\tau)^T \right\} \mathbf{B}_w^T (e^{\mathbf{A}(t_n-\tau)})^T d\tau dt \\ &= \int_{t_{n-1}}^{t_n} \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_w R_{ww}(t-\tau) \mathbf{B}_w^T (e^{\mathbf{A}(t_n-\tau)})^T d\tau dt \\ &= \int_{t_{n-1}}^{t_n} \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-t)} \mathbf{B}_w \boldsymbol{\Sigma}_w \delta(t-\tau) \mathbf{B}_w^T (e^{\mathbf{A}(t_n-\tau)})^T d\tau dt \\ &= \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n-\tau)} \mathbf{B}_w \boldsymbol{\Sigma}_w \mathbf{B}_w^T e^{\mathbf{A}^T(t_n-\tau)} d\tau\end{aligned}$$

Properties of the Process Noise

- Process noise:

$$\mathbf{q}_n \triangleq \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_w \mathbf{w}(t) dt$$

- Mean and covariance:

$$E\{\mathbf{q}_n\} = 0$$

$$\text{Cov}\{\mathbf{q}_n\} = \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - \tau)} \mathbf{B}_w \boldsymbol{\Sigma}_w \mathbf{B}_w^T e^{\mathbf{A}^T(t_n - \tau)} d\tau \triangleq \mathbf{Q}_n$$

- Distribution:

$$\mathbf{q}_n \sim \mathcal{N}(0, \mathbf{Q}_n)$$

Discretized Stochastic Linear Dynamic Model

- Discretized stochastic dynamic model:

$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{q}_n$$

where:

$$\mathbf{F}_n \triangleq e^{\mathbf{A}(t_n - t_{n-1})}$$

$$\mathbf{q}_n \sim \mathcal{N}(0, \mathbf{Q}_n)$$

$$\mathbf{Q}_n = \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - \tau)} \mathbf{B}_w \boldsymbol{\Sigma}_w \mathbf{B}_w^\top e^{\mathbf{A}^\top(t_n - \tau)} d\tau$$

The discretized stochastic dynamic model is completely equivalent to the continuous-time model

Example: 1D Wiener Velocity Model (1/3)

- Dynamic model:

$$\begin{bmatrix} \dot{p}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w(t)$$

with white noise process $w(t)$ and $R_{ww}(\tau) = \sigma_w^2 \delta(\tau)$

- Process noise covariance:

$$\mathbf{Q}_n = \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - \tau)} \mathbf{B}_w \boldsymbol{\Sigma}_w \mathbf{B}_w^\top e^{\mathbf{A}^\top(t_n - \tau)} d\tau$$

- Recall:

$$e^{\mathbf{A}(t_n - \tau)} = \begin{bmatrix} 1 & t_n - \tau \\ 0 & 1 \end{bmatrix}$$

$$e^{\mathbf{A}(t_n - \tau)} \mathbf{B}_w = \begin{bmatrix} 1 & t_n - \tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} t_n - \tau \\ 1 \end{bmatrix}$$

Example: 1D Wiener Velocity Model (2/3)

- Process noise covariance:

$$\begin{aligned}\mathbf{Q}_n &= \int_{t_{n-1}}^{t_n} \begin{bmatrix} t_n - \tau \\ 1 \end{bmatrix} \sigma_w^2 \begin{bmatrix} t_n - \tau \\ 1 \end{bmatrix}^\top d\tau \\ &= \sigma_w^2 \int_{t_{n-1}}^{t_n} \begin{bmatrix} t_n - \tau \\ 1 \end{bmatrix} \begin{bmatrix} t_n - \tau & 1 \end{bmatrix} d\tau \\ &= \sigma_w^2 \int_{t_{n-1}}^{t_n} \begin{bmatrix} (t_n - \tau)^2 & t_n - \tau \\ t_n - \tau & 1 \end{bmatrix} d\tau \\ &= \sigma_w^2 \left[\begin{array}{cc} -\frac{1}{3}(t_n - \tau)^3 & -\frac{1}{2}(t_n - \tau)^2 \\ -\frac{1}{2}(t_n - \tau)^2 & \tau \end{array} \right]_{\tau=t_{n-1}}^{t_n} \\ &= \sigma_w^2 \begin{bmatrix} \frac{(\Delta t)^3}{3} & \frac{(\Delta t)^2}{2} \\ \frac{(\Delta t)^2}{2} & \Delta t \end{bmatrix}\end{aligned}$$

Example: 1D Wiener Velocity Model (3/3)

- Dynamic model:

$$\begin{bmatrix} \dot{p}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w(t)$$

- Discretized model:

$$\begin{bmatrix} p_n \\ v_n \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{n-1} \\ v_{n-1} \end{bmatrix} + \mathbf{q}_n$$

with $\mathbf{q}_n \sim \mathcal{N}(0, \mathbf{Q}_n)$ and

$$\mathbf{Q}_n = \sigma_w^2 \begin{bmatrix} \frac{(\Delta t)^3}{3} & \frac{(\Delta t)^2}{2} \\ \frac{(\Delta t)^2}{2} & \Delta t \end{bmatrix}$$

Discretization of Nonlinear Dynamic Models

- Objective: Discretization of nonlinear models

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_u(\mathbf{x}(t))\mathbf{u}(t)$$

and

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

- Problem: In most cases, no exact approach exists
- A few possible approaches:
 - Linearization of the nonlinear model followed by discretization
 - Approximation of the derivative (integral)
 - Exact integration (of at least the dynamics)
 - & many more...

Linearization of Nonlinear Models

- Nonlinear dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_u(\mathbf{x}(t))\mathbf{u}(t)$$

- 1st order Taylor series approximation of $\mathbf{f}(\mathbf{x}(t))$ around $\mathbf{x}(t) = \mathbf{x}(t_{n-1})$:

$$\mathbf{f}(\mathbf{x}(t)) \approx \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{A}_x(\mathbf{x}(t) - \mathbf{x}_{n-1})$$

- Approximation of the ODE:

$$\dot{\mathbf{x}}(t) \approx \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{A}_x(\mathbf{x}(t) - \mathbf{x}_{n-1}) + \mathbf{B}_u\mathbf{u}(t)$$

Discretization of Linearized Models (1/2)

- Approximation of the ODE:

$$\dot{\mathbf{x}}(t) \approx \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{A}_x(\mathbf{x}(t) - \mathbf{x}_{n-1}) + \mathbf{B}_u \mathbf{u}(t)$$

- Rewritten approximation of the ODE

$$\dot{\mathbf{x}}(t) \approx \mathbf{A}_x \mathbf{x}(t) + \mathbf{f}(\mathbf{x}_{n-1}) - \mathbf{A}_x \mathbf{x}_{n-1} + \mathbf{B}_u \mathbf{u}(t)$$

- Solution of the approximation:

$$\begin{aligned}\mathbf{x}_n \approx & e^{\mathbf{A}_x \Delta t} \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} dt \mathbf{f}(\mathbf{x}_{n-1}) \\ & - \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} dt \mathbf{A}_x \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} \mathbf{B}_u \mathbf{u}(t) dt\end{aligned}$$

Discretization of Linearized Models (2/2)

- Solution of the approximation:

$$\begin{aligned}\mathbf{x}_n \approx & e^{\mathbf{A}_x \Delta t} \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} dt \mathbf{f}(\mathbf{x}_{n-1}) \\ & - \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} dt \mathbf{A}_x \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} \mathbf{B}_u \mathbf{u}(t) dt\end{aligned}$$

- Simplified solution:

$$\mathbf{x}_n \approx \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} dt \mathbf{f}(\mathbf{x}_{n-1}) + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} \mathbf{B}_u \mathbf{u}(t) dt$$

Discretization of Linearized Models (Stochastic)

- Stochastic nonlinear model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

- Discretization is the same as for the ODE model:

$$\begin{aligned}\mathbf{x}_n &\approx \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} dt \mathbf{f}(\mathbf{x}_{n-1}) + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} \mathbf{B}_w \mathbf{w}(t) dt \\ &= \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} dt \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n\end{aligned}$$

with

$$\mathbf{q}_n \sim \mathcal{N}(0, \mathbf{Q}_n),$$

$$\mathbf{Q}_n \approx \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-\tau)} \mathbf{B}_w \boldsymbol{\Sigma}_w \mathbf{B}_w^T e^{\mathbf{A}_x^T(t_n-\tau)} d\tau$$

Properties of the Discretization

- Stochastic nonlinear model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

- Linearized model:

$$\dot{\mathbf{x}}(t) \approx \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{A}_x(x(t) - x_{n-1}) + \mathbf{B}_w\mathbf{w}(t)$$

- Discretized model:

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n-t)} dt \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n$$

- Integration is exact, model is not
- Discretization is not exact
- Linearization is local, may cause problems

Example: Quasi-Constant Turn Model (1/5)

- Model:

$$\begin{bmatrix} \dot{p}^x(t) \\ \dot{p}^y(t) \\ \dot{v}(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} v(t) \cos(\varphi(t)) \\ v(t) \sin(\varphi(t)) \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{w}(t)$$

- Jacobian of $\mathbf{f}(\mathbf{x}(t))$:

$$\begin{aligned} \mathbf{A}_x &= \begin{bmatrix} 0 & 0 & \cos(\varphi(t)) & -v(t) \sin(\varphi(t)) \\ 0 & 0 & \sin(\varphi(t)) & v(t) \cos(\varphi(t)) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & \cos(\varphi_{n-1}) & -v_{n-1} \sin(\varphi_{n-1}) \\ 0 & 0 & \sin(\varphi_{n-1}) & v_{n-1} \cos(\varphi_{n-1}) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned}$$

Example: Quasi-Constant Turn Model (2/5)

- Powers of \mathbf{A}_x :

$$\mathbf{A}_x^0 = \mathbf{I}$$

$$\mathbf{A}_x^1 = \mathbf{A}_x$$

$$\mathbf{A}_x^2 = \mathbf{0}$$

- Matrix exponential:

$$e^{\mathbf{A}_x(t_n - t)} = \mathbf{I} + \mathbf{A}_x(t_n - t)$$

$$= \begin{bmatrix} 1 & 0 & \cos(\varphi_{n-1})(t_n - t) & -v_{n-1} \sin(\varphi_{n-1})(t_n - t) \\ 0 & 1 & \sin(\varphi_{n-1})(t_n - t) & v_{n-1} \cos(\varphi_{n-1})(t_n - t) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example: Quasi-Constant Turn Model (3/5)

- Integral:

$$\begin{aligned}& \int_{t_{n-1}}^{t_n} \begin{bmatrix} 1 & 0 & \cos(\varphi_{n-1})(t_n - t) & -v_{n-1} \sin(\varphi_{n-1})(t_n - t) \\ 0 & 1 & \sin(\varphi_{n-1})(t_n - t) & v_{n-1} \cos(\varphi_{n-1})(t_n - t) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} dt \\&= \left[\begin{array}{cccc} t & 0 & -\frac{(t_n-t)^2}{2} \cos(\varphi_{n-1}) & \frac{(t_n-t)^2}{2} v_{n-1} \sin(\varphi_{n-1}) \\ 0 & t & -\frac{(t_n-t)^2}{2} \sin(\varphi_{n-1}) & -\frac{(t_n-t)^2}{2} v_{n-1} \cos(\varphi_{n-1}) \\ 0 & 0 & t & 0 \\ 0 & 0 & 0 & t \end{array} \right]_{t=t_{n-1}}^{t_n} \\&= \left[\begin{array}{cccc} \Delta t & 0 & \frac{(\Delta t)^2}{2} \cos(\varphi_{n-1}) & -\frac{(\Delta t)^2}{2} v_{n-1} \sin(\varphi_{n-1}) \\ 0 & \Delta t & \frac{(\Delta t)^2}{2} \sin(\varphi_{n-1}) & \frac{(\Delta t)^2}{2} v_{n-1} \cos(\varphi_{n-1}) \\ 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & \Delta t \end{array} \right]\end{aligned}$$

Example: Quasi-Constant Turn Model (4/5)

- Discretized model:

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n - t)} dt \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n$$

- Second term:

$$\begin{bmatrix} \Delta t & 0 & \frac{(\Delta t)^2}{2} \cos(\varphi_{n-1}) & -\frac{(\Delta t)^2}{2} v_{n-1} \sin(\varphi_{n-1}) \\ 0 & \Delta t & \frac{(\Delta t)^2}{2} \sin(\varphi_{n-1}) & \frac{(\Delta t)^2}{2} v_{n-1} \cos(\varphi_{n-1}) \\ 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & \Delta t \end{bmatrix} \begin{bmatrix} v_{n-1} \cos(\varphi_{n-1}) \\ v_{n-1} \sin(\varphi_{n-1}) \\ 0 \\ 0 \end{bmatrix}$$
$$= \begin{bmatrix} \Delta t v_{n-1} \cos(\varphi_{n-1}) \\ \Delta t v_{n-1} \sin(\varphi_{n-1}) \\ 0 \\ 0 \end{bmatrix}$$

Example: Quasi-Constant Turn Model (5/5)

- Discretized model:

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n - t)} dt \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n$$

- Discretization of Linearized Model:

$$\begin{bmatrix} p_n^x \\ p_n^y \\ v_n \\ \varphi_n \end{bmatrix} = \begin{bmatrix} p_{n-1}^x \\ p_{n-1}^y \\ v_{n-1} \\ \varphi_{n-1} \end{bmatrix} + \begin{bmatrix} \Delta t v_{n-1} \cos(\varphi_{n-1}) \\ \Delta t v_{n-1} \sin(\varphi_{n-1}) \\ 0 \\ 0 \end{bmatrix} + \mathbf{q}_n$$

- What about \mathbf{Q}_n ?

$$\mathbf{Q}_n \approx \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n - \tau)} \mathbf{B}_w \boldsymbol{\Sigma}_w \mathbf{B}_w^T e^{\mathbf{A}_x^T(t_n - \tau)} d\tau$$

Euler Approximation

- Dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_u(\mathbf{x}(t))\mathbf{u}(t)$$

- Integral equation:

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} \mathbf{f}(\mathbf{x}(t))dt + \int_{t_{n-1}}^{t_n} \mathbf{B}_u(\mathbf{x}(t))\mathbf{u}(t)dt$$

- Idea: Approximate the integral rather than the model
- Euler approximation:

$$\mathbf{x}_n \approx \mathbf{x}_{n-1} + \Delta t \mathbf{f}(\mathbf{x}_{n-1}) + \Delta t \mathbf{B}_u(\mathbf{x}_{n-1})\mathbf{u}_{n-1}.$$

Euler–Maruyama Discretization (1)

- Stochastic dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

- Integral representation:

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} \mathbf{f}(\mathbf{x}(t))dt + \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)dt$$

- Process noise definition:

$$\mathbf{q}_n \triangleq \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)dt$$

Mean of the Process Noise

- Process noise:

$$\mathbf{q}_n \triangleq \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t)) \mathbf{w}(t) dt$$

- Mean:

$$\begin{aligned}\mathbb{E}\{\mathbf{q}_n\} &= \mathbb{E} \left\{ \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t)) \mathbf{w}(t) dt \right\} \\ &= \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t)) \mathbb{E}\{\mathbf{w}(t)\} dt \\ &= 0\end{aligned}$$

Covariance of the Process Noise (1/2)

- Process noise:

$$\mathbf{q}_n \triangleq \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t)) \mathbf{w}(t) dt$$

- Covariance:

$$\begin{aligned}\text{Cov}\{\mathbf{q}_n\} &= E \left\{ \left(\int_{t_{n-1}}^{t_n} \mathbf{B}_w \mathbf{w}(t) dt \right) \left(\int_{t_{n-1}}^{t_n} \mathbf{B}_w \mathbf{w}(\tau) d\tau \right)^T \right\} \\ &= \int_{t_{n-1}}^{t_n} \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t)) E\{\mathbf{w}(t)\mathbf{w}(\tau)^T\} \mathbf{B}_w(\mathbf{x}(t))^T d\tau dt \\ &= \int_{t_{n-1}}^{t_n} \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t)) \boldsymbol{\Sigma}_w \delta(t - \tau) \mathbf{B}_w(\mathbf{x}(t))^T d\tau dt \\ &= \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t)) \boldsymbol{\Sigma}_w \mathbf{B}_w^T(\mathbf{x}(t)) dt\end{aligned}$$

Covariance of the Process Noise (2/2)

- Covariance:

$$\text{Cov}\{\mathbf{q}_n\} = \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t)) \boldsymbol{\Sigma}_w \mathbf{B}_w(\mathbf{x}(t))^T dt$$

- Rectangle approximation of the integral:

$$\begin{aligned}\text{Cov}\{\mathbf{q}_n\} &= \int_{t_{n-1}}^{t_n} \mathbf{B}_w(\mathbf{x}(t)) \boldsymbol{\Sigma}_w \mathbf{B}_w(\mathbf{x}(t))^T dt \\ &\approx \mathbf{B}_w(\mathbf{x}_{n-1}) \boldsymbol{\Sigma}_w \mathbf{B}_w(\mathbf{x}_{n-1})^T (t_n - t_{n-1}) \\ &= \Delta t \mathbf{B}_w(\mathbf{x}_{n-1}) \boldsymbol{\Sigma}_w \mathbf{B}_w(\mathbf{x}_{n-1})^T \\ &\triangleq \mathbf{Q}_n\end{aligned}$$

Euler–Maruyama Discretization (2)

- Dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

- Euler–Maruyama discretization:

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \Delta t \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n$$

with $\mathbf{q}_n \sim \mathcal{N}(0, \mathbf{Q}_n)$, $\mathbf{Q}_n \approx \Delta t \mathbf{B}_w(\mathbf{x}_{n-1}) \boldsymbol{\Sigma}_w \mathbf{B}_w(\mathbf{x}_{n-1})^\top$

- ... or equivalently:

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \Delta t \mathbf{f}(\mathbf{x}_{n-1}) + \sqrt{\Delta t} \mathbf{B}_w(\mathbf{x}_{n-1}) \mathbf{q}_n$$

with $\mathbf{q}_n \sim \mathcal{N}(0, \boldsymbol{\Sigma}_w)$

- Discretization is not exact

Summary (1/3)

- The discretization of the linear ODE model

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u\mathbf{u}(t)$$

is

$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{L}_n \mathbf{u}_{n-1}$$

$$\mathbf{F}_n \triangleq e^{\mathbf{A}(t_n - t_{n-1})}, \quad \mathbf{L}_n \triangleq \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_u dt$$

- The discretization of the linear SDE model

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_w\mathbf{w}(t)$$

is

$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{q}_n, \quad \mathbf{q}_n \sim \mathcal{N}(0, \mathbf{Q}_n)$$

$$\mathbf{Q}_n = \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - \tau)} \mathbf{B}_w \boldsymbol{\Sigma}_w \mathbf{B}_w^\top e^{\mathbf{A}^\top(t_n - \tau)} d\tau$$

Summary (2/3)

- Nonlinear dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

- Discretization of the linearized model:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w \mathbf{w}(t) \\ &\approx \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{A}_x(\mathbf{x}(t) - \mathbf{x}_{n-1}) + \mathbf{B}_w \mathbf{w}(t) \\ &\quad \Downarrow \\ \mathbf{x}_n &= \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n - \tau)} dt \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n\end{aligned}$$

with

$$\mathbf{q}_n \sim \mathcal{N}(0, \mathbf{Q}_n), \quad \mathbf{Q}_n \approx \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n - \tau)} \mathbf{B}_w \boldsymbol{\Sigma}_w \mathbf{B}_w^\top e^{\mathbf{A}_x^\top(t_n - \tau)} d\tau$$

Summary (3/3)

- Euler–Maruyama discretization:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t) \\ &\Downarrow \\ \mathbf{x}_n &= \mathbf{x}_{n-1} + \Delta t \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n\end{aligned}$$

with

$$\begin{aligned}\mathbf{q}_n &\sim \mathcal{N}(0, \mathbf{Q}_n), \\ \mathbf{Q}_n &\approx \Delta t \mathbf{B}_w(\mathbf{x}_{n-1}) \boldsymbol{\Sigma}_w \mathbf{B}_w(\mathbf{x}_{n-1})^\top.\end{aligned}$$



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Filtering Problem and Kalman Filtering

Simo Särkkä

Aalto University

November 13, 2020

Contents

- 1 Intended Learning Outcomes and Recap
- 2 Kalman Filter Theory
- 3 Kalman Filter Practice
- 4 Summary

Intended Learning Outcomes

After this lecture, you will be able to:

- explain the relationship between the **dynamic model**, **measurement model**, and the **filtering methodology**,
- describe and employ the **Kalman filter** for linear state-space models.

Recap (1/3)

- The discretization of the linear ODE model

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_u\mathbf{u}(t)$$

is

$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{L}_n \mathbf{u}_{n-1}$$

$$\mathbf{F}_n \triangleq e^{\mathbf{A}(t_n - t_{n-1})}, \quad \mathbf{L}_n \triangleq \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - t)} \mathbf{B}_u dt$$

- The discretization of the linear SDE model

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_w\mathbf{w}(t)$$

is

$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{q}_n, \quad \mathbf{q}_n \sim \mathcal{N}(0, \mathbf{Q}_n)$$

$$\mathbf{Q}_n = \int_{t_{n-1}}^{t_n} e^{\mathbf{A}(t_n - \tau)} \mathbf{B}_w \boldsymbol{\Sigma}_w \mathbf{B}_w^\top e^{\mathbf{A}^\top(t_n - \tau)} d\tau$$

Recap (2/3)

- Nonlinear dynamic model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$

- Discretization of the linearized model:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w \mathbf{w}(t) \\ &\approx \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{A}_x(\mathbf{x}(t) - \mathbf{x}_{n-1}) + \mathbf{B}_w \mathbf{w}(t) \\ &\quad \Downarrow \\ \mathbf{x}_n &= \mathbf{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n - \tau)} dt \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n\end{aligned}$$

with

$$\mathbf{q}_n \sim \mathcal{N}(0, \mathbf{Q}_n), \quad \mathbf{Q}_n \approx \int_{t_{n-1}}^{t_n} e^{\mathbf{A}_x(t_n - \tau)} \mathbf{B}_w \boldsymbol{\Sigma}_w \mathbf{B}_w^\top e^{\mathbf{A}_x^\top(t_n - \tau)} d\tau$$

Recap (3/3)

- Euler–Maruyama discretization:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{B}_w(\mathbf{x}(t))\mathbf{w}(t)$$
$$\Downarrow$$

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \Delta t \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n$$

with

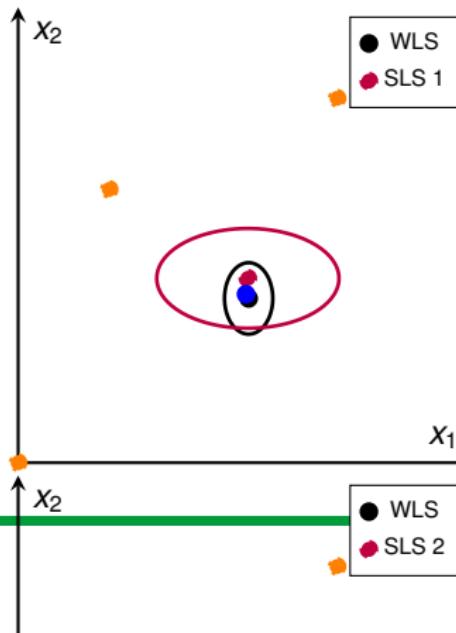
$$\mathbf{q}_n \sim \mathcal{N}(0, \mathbf{Q}_n),$$

$$\mathbf{Q}_n \approx \Delta t \mathbf{B}_w(\mathbf{x}_{n-1}) \boldsymbol{\Sigma}_w \mathbf{B}_w(\mathbf{x}_{n-1})^\top.$$

Recall: Car Localization with Sequential Least Squares

- Goal: Estimate the position $\mathbf{x} = [p^x \quad p^y]^T$
- Measurements: Noisy position measurements $n = 1, 2, \dots$:

$$\mathbf{y}_n = \mathbf{G}_n \mathbf{x}_n + \mathbf{r}_n$$



The Filtering Approach

- We can also allow the target to move between the measurements.
 - This movement can be described by a stochastic dynamic model.
- The measurement can be handled as in sequential least squares.
- Filter iterates the following two steps for all points in time:
 - ➊ **Prediction:** Predict the current state using the dynamic model (also called *time update*)
 - ➋ **Measurement update:** Estimate the current state using the prediction and the new measurement

The Filtering Approach: Prediction

- Objective: Predict the current state \mathbf{x}_n at t_n given all previous data $\mathbf{y}_{1:n-1} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{n-1}\}$
- Done by solving the dynamic model equation starting from the mean and covariance at the previous step.
- Notation:
 - $\hat{\mathbf{x}}_{n|n-1}$: Denotes the predicted value of \mathbf{x}_n given the measurements $\mathbf{y}_{1:n-1} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{n-1}\}$
 - $\mathbf{P}_{n|n-1}$: Denotes the covariance of the predicted \mathbf{x}_n
- Prediction adds uncertainty

The Filtering Approach: Measurement Update

- Objective: Estimate the current value of \mathbf{x}_n given the new measurement \mathbf{y}_n , taking the prediction into account
- Done by solving the regularized least squares problem with the predicted result as the regularization term.
- Notation:
 - $\hat{\mathbf{x}}_{n|n}$: Denotes the estimated value at t_n , given the measurements $\mathbf{y}_{1:n} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$
 - $\mathbf{P}_{n|n}$: Denotes the covariance at t_n
- The measurement update reduces uncertainty

Linear State-Space Model

- Linear state-space model:

$$\dot{\mathbf{x}}(t) = \mathbf{Ax}(t) + \mathbf{Bw}(t)$$

$$\mathbf{y}_n = \mathbf{G}_n \mathbf{x}(t_n) + \mathbf{r}_n$$

- Discrete-time equivalent:

$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{q}_n$$

$$\mathbf{y}_n = \mathbf{G}_n \mathbf{x}_n + \mathbf{r}_n$$

with

$$E\{\mathbf{q}_n\} = 0, \text{ Cov}\{\mathbf{q}_n\} = \mathbf{Q}_n,$$

$$E\{\mathbf{r}_n\} = 0, \text{ Cov}\{\mathbf{r}_n\} = \mathbf{R}_n$$

- Initial conditions:

$$E\{\mathbf{x}_0\} = \mathbf{m}_0$$

$$\text{Cov}\{\mathbf{x}_0\} = \mathbf{P}_0$$

Linear Model: Prediction (1/2)

- Linear dynamic model:

$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{q}_n, \quad \text{Cov}\{\mathbf{q}_n\} = \mathbf{Q}_n$$

- Given:

$$\begin{aligned} E\{\mathbf{x}_{n-1} \mid \mathbf{y}_{1:n-1}\} &= \hat{\mathbf{x}}_{n-1|n-1}, \\ \text{Cov}\{\mathbf{x}_{n-1} \mid \mathbf{y}_{1:n-1}\} &= \mathbf{P}_{n-1|n-1}. \end{aligned}$$

- Predicted mean:

$$\begin{aligned} \hat{\mathbf{x}}_{n|n-1} &= E\{\mathbf{x}_n \mid \mathbf{y}_{1:n-1}\} \\ &= \mathbf{F}_n \hat{\mathbf{x}}_{n-1|n-1} \end{aligned}$$

Linear Model: Prediction (2/2)

- Linear dynamic model:

$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{q}_n, \quad \text{Cov}\{\mathbf{q}_n\} = \mathbf{Q}_n$$

- Given:

$$\mathbb{E}\{\mathbf{x}_{n-1} \mid \mathbf{y}_{1:n-1}\} = \hat{\mathbf{x}}_{n-1|n-1},$$

$$\text{Cov}\{\mathbf{x}_{n-1} \mid \mathbf{y}_{1:n-1}\} = \mathbf{P}_{n-1|n-1}.$$

- Covariance:

$$\mathbf{P}_{n|n-1} = \text{Cov}\{\mathbf{x}_n \mid \mathbf{y}_{1:n-1}\}$$

$$= \mathbb{E}\{(\mathbf{x}_n - \mathbb{E}\{\mathbf{x}_n \mid \mathbf{y}_{1:n-1}\})(\mathbf{x}_n - \mathbb{E}\{\mathbf{x}_n \mid \mathbf{y}_{1:n-1}\})^T \mid \mathbf{y}_{1:n-1}\}$$

$$= \mathbf{F}_n \mathbf{P}_{n-1|n-1} \mathbf{F}_n^T + \mathbf{Q}_n$$

Linear Model: Measurement Update (1/3)

- Assume that the prediction yields the **prior** knowledge $\hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}$
- \mathbf{y}_n provides the new information of the state
- We can use **regularized least squares** to estimate \mathbf{x}_n :

$$\begin{aligned} J_{\text{ReLS}}(\mathbf{x}_n) = & (\mathbf{y}_n - \mathbf{G}_n \mathbf{x}_n)^T \mathbf{R}_n^{-1} (\mathbf{y}_n - \mathbf{G}_n \mathbf{x}_n) \\ & + (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^T \mathbf{P}_{n|n-1}^{-1} (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) \end{aligned}$$

and solve

$$\hat{\mathbf{x}}_{n|n} = \underset{\mathbf{x}_n}{\operatorname{argmin}} J_{\text{ReLS}}(\mathbf{x}_n)$$

Linear Model: Measurement Update (2/3)

- Regularized least squares problem:

$$\begin{aligned}\hat{\mathbf{x}}_{n|n} = \operatorname{argmin}_{\mathbf{x}_n} & (\mathbf{y}_n - \mathbf{G}_n \mathbf{x}_n)^T \mathbf{R}_n^{-1} (\mathbf{y}_n - \mathbf{G}_n \mathbf{x}_n) \\ & + (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^T \mathbf{P}_{n|n-1}^{-1} (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})\end{aligned}$$

- Solution (see Lecture 3 / Chapter 3.4):

$$\begin{aligned}\mathbf{K}_n &= \mathbf{P}_{n|n-1} \mathbf{G}_n^T (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n^T + \mathbf{R}_n)^{-1} \\ \hat{\mathbf{x}}_{n|n} &= \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{G}_n \hat{\mathbf{x}}_{n|n-1})\end{aligned}$$

- Covariance of $\hat{\mathbf{x}}_{n|n}$:

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n^T + \mathbf{R}_n) \mathbf{K}_n^T$$

- \mathbf{K}_n is called the *Kalman gain*

Linear Model: Measurement Update (3/3)

- Measurement update:

$$\mathbf{K}_n = \mathbf{P}_{n|n-1} \mathbf{G}_n^T (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n^T + \mathbf{R}_n)^{-1}$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{G}_n \hat{\mathbf{x}}_{n|n-1})$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n^T + \mathbf{R}_n) \mathbf{K}_n^T$$

- It can be shown that it holds that:

$$E\{\mathbf{x}_n | \mathbf{y}_{1:n}\} = \hat{\mathbf{x}}_{n|n}$$

$$\text{Cov}\{\mathbf{x}_n | \mathbf{y}_{1:n}\} = \mathbf{P}_{n|n}$$

Linear Model: Summary

- Prediction:

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{F}_n \hat{\mathbf{x}}_{n-1|n-1}$$

$$\mathbf{P}_{n|n-1} = \mathbf{F}_n \mathbf{P}_{n-1|n-1} \mathbf{F}_n^T + \mathbf{Q}_n$$

- Measurement update:

$$\mathbf{K}_n = \mathbf{P}_{n|n-1} \mathbf{G}_n^T (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n^T + \mathbf{R}_n)^{-1}$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{G}_n \hat{\mathbf{x}}_{n|n-1})$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n^T + \mathbf{R}_n) \mathbf{K}_n^T$$

- Initialization of the recursion:

$$\hat{\mathbf{x}}_{0|0} = \mathbf{m}_0$$

$$\mathbf{P}_{0|0} = \mathbf{P}_0$$

The Kalman Filter

Algorithm 1 Kalman Filter

1: Initialize $\hat{\mathbf{x}}_{0|0} = \mathbf{m}_0$, $\mathbf{P}_{0|0} = \mathbf{P}_0$

2: **for** $n = 1, 2, \dots$ **do**

3: Prediction (time update):

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{F}_n \hat{\mathbf{x}}_{n-1|n-1}$$

$$\mathbf{P}_{n|n-1} = \mathbf{F}_n \mathbf{P}_{n-1|n-1} \mathbf{F}_n^T + \mathbf{Q}_n$$

4: Measurement update:

$$\mathbf{K}_n = \mathbf{P}_{n|n-1} \mathbf{G}_n^T (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n + \mathbf{R}_n)^{-1}$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{G}_n \hat{\mathbf{x}}_{n|n-1})$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n + \mathbf{R}_n) \mathbf{K}_n^T$$

5: **end for**

Example: Car Localization (1/3)

- Goal: Estimate the kinematic state at each time t_n
- Dynamic model: 2D Wiener velocity model:

$$\begin{bmatrix} \dot{p}^x(t) \\ \dot{p}^y(t) \\ \dot{v}^x(t) \\ \dot{v}^y(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p^x(t) \\ p^y(t) \\ v^x(t) \\ v^y(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} w_1(t) \\ w_2(t) \end{bmatrix}$$

- Measurements: Noisy position ($\mathbf{G}_n = [\mathbf{I} \quad \mathbf{0}]$):

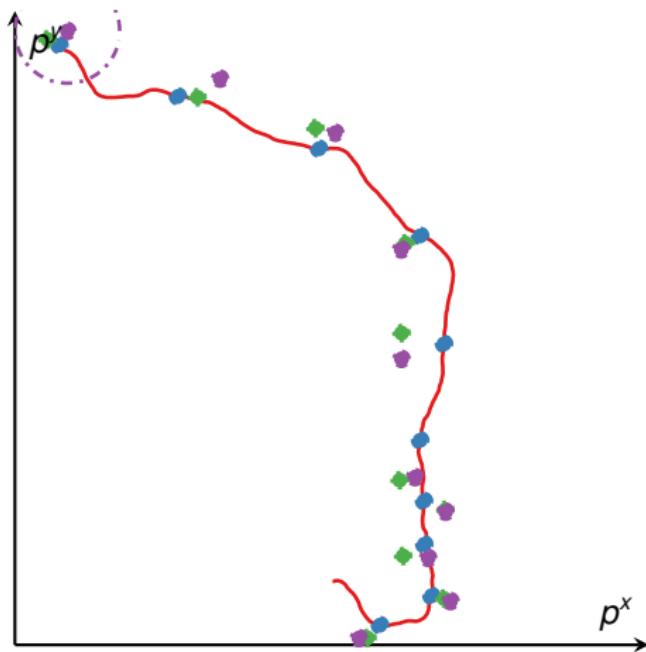
$$\mathbf{y}_n = \mathbf{G}_n \mathbf{x}_n + \mathbf{r}_n$$

- Discrete-time linear state-space model:

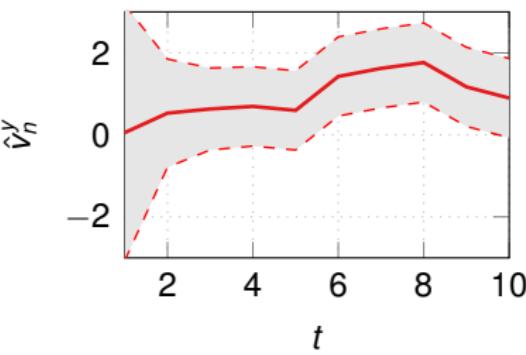
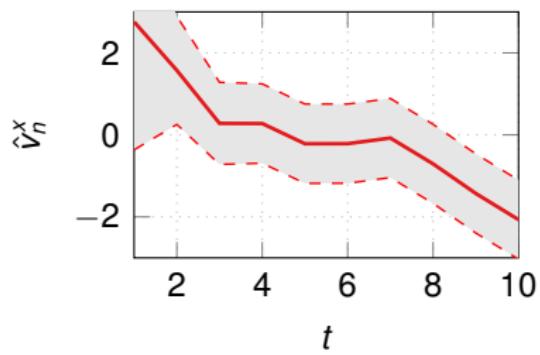
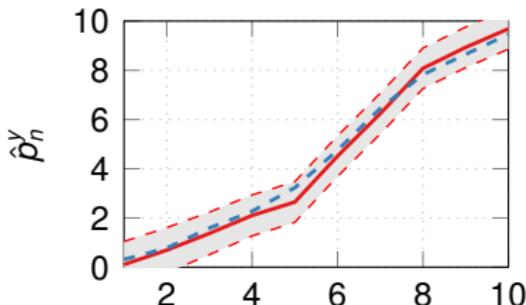
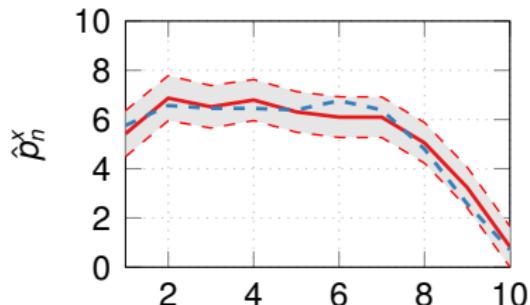
$$\mathbf{x}_n = \mathbf{F}_n \mathbf{x}_{n-1} + \mathbf{q}_n$$

$$\mathbf{y}_n = \mathbf{G}_n \mathbf{x}_n + \mathbf{r}_n$$

Example: Car Localization (2/3)



Example: Car Localization (3/3)



Performance Evaluation

A few questions:

- Why not just use the measurements \mathbf{y}_n as the position estimate ($\hat{\mathbf{p}}_n = \mathbf{y}_n$)?
- How should we assess the performance of the algorithm?
- One possible criterion: The root mean squared error (RMSE):

$$e_{\text{RMSE}} = \sqrt{\frac{1}{N} \sum_{n=1}^N (\hat{\mathbf{x}}_{n|n} - \mathbf{x}_n)^T (\hat{\mathbf{x}}_{n|n} - \mathbf{x}_n)}$$

Example: Car Localization

- RMSE for the primitive approach ($\hat{\mathbf{p}}_n = \mathbf{y}_n$):

$$e_{\text{RMSE}} = 0.41$$

- RMSE for Kalman filter:

$$e_{\text{RMSE}} = 0.29$$

- The prior knowledge imposed by the dynamic model significantly improves performance!

Measurement Update: Some Observations

- Measurement update:

$$\mathbf{K}_n = \mathbf{P}_{n|n-1} \mathbf{G}_n^T (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n + \mathbf{R}_n)^{-1}$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{G}_n \hat{\mathbf{x}}_{n|n-1})$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n + \mathbf{R}_n) \mathbf{K}_n^T$$

- Prediction of the output and covariances:

$$E\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\} = \mathbf{G}_n \hat{\mathbf{x}}_{n|n-1}$$

$$\text{Cov}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\} = \mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n^T + \mathbf{R}_n$$

$$\text{Cov}\{\mathbf{x}_n, \mathbf{y}_n | \mathbf{y}_{1:n-1}\} = \mathbf{P}_{n|n-1} \mathbf{G}_n^T$$

Summary

- The filtering approach iterates between two steps:
 - 1 Prediction: $\hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1} \Rightarrow \hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}$
 - 2 Measurement update: $\hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1} \Rightarrow \hat{\mathbf{x}}_{n|n}, \mathbf{P}_{n|n}$
- The **Kalman filter** is the optimal filter for linear state-space models
 - 1 Prediction:

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{F}_n \hat{\mathbf{x}}_{n|n-1}$$

$$\mathbf{P}_{n|n-1} = \mathbf{F}_n \mathbf{P}_{n|n-1} \mathbf{F}_n^T + \mathbf{Q}_n$$

- 2 Measurement update:

$$\mathbf{K}_n = \mathbf{P}_{n|n-1} \mathbf{G}_n^T (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n + \mathbf{R}_n)^{-1}$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{G}_n \hat{\mathbf{x}}_{n|n-1})$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n + \mathbf{R}_n) \mathbf{K}_n^T$$



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Extended and Unscented Kalman Filtering

Simo Särkkä

Aalto University

November 20, 2020

Contents

- 1 Intended Learning Outcomes and Recap
- 2 Nonlinear State-Space Models and Extended Kalman Filter
- 3 Unscented Kalman Filter
- 4 Summary

Intended Learning Outcomes

After this lecture, you will be able to:

- recognize the challenges for filtering in nonlinear state-space models,
- describe and employ the extended and unscented Kalman filters for nonlinear state-space models

Recap: Filtering and the Kalman Filter

- The filtering approach iterates between two steps:
 - Prediction: $\hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1} \Rightarrow \hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1}$
 - Measurement update: $\hat{\mathbf{x}}_{n|n-1}, \mathbf{P}_{n|n-1} \Rightarrow \hat{\mathbf{x}}_{n|n}, \mathbf{P}_{n|n}$
- The **Kalman filter** is the optimal filter for linear state-space models
 - Prediction:

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{F}_n \hat{\mathbf{x}}_{n|n-1}$$

$$\mathbf{P}_{n|n-1} = \mathbf{F}_n \mathbf{P}_{n-1|n-1} \mathbf{F}_n^T + \mathbf{Q}_n$$

- Measurement update:

$$\mathbf{K}_n = \mathbf{P}_{n|n-1} \mathbf{G}_n^T (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n + \mathbf{R}_n)^{-1}$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{G}_n \hat{\mathbf{x}}_{n|n-1})$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_n \mathbf{P}_{n|n-1} \mathbf{G}_n + \mathbf{R}_n) \mathbf{K}_n^T$$

Discrete-Time Nonlinear State-Space Model

- Discrete-time nonlinear state-space model:

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n$$

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n$$

- Process and measurement noises (\mathbf{q}_n and \mathbf{r}_n):

$$E\{\mathbf{q}_n\} = 0, \text{ Cov}\{\mathbf{q}_n\} = \mathbf{Q}_n$$

$$E\{\mathbf{r}_n\} = 0, \text{ Cov}\{\mathbf{r}_n\} = \mathbf{R}_n$$

- Initial conditions:

$$E\{\mathbf{x}_0\} = \mathbf{m}_0, \text{ Cov}\{\mathbf{x}_0\} = \mathbf{P}_0$$

Filtering for Nonlinear Models

- For most nonlinear models, exact prediction and/or update steps can not be found
- Example: Prediction for general nonlinear model

$$\hat{\mathbf{x}}_{n|n-1} = E\{\mathbf{x}_n \mid \mathbf{y}_{1:n-1}\}$$

Approximations to the exact solutions are required!

Linearized Model: Prediction (1/2)

- State estimate from t_{n-1} : $\hat{\mathbf{x}}_{n-1|n-1}$, $\mathbf{P}_{n-1|n-1}$
- Linearization around $\hat{\mathbf{x}}_{n-1|n-1}$ (dynamic model):

$$\begin{aligned}\mathbf{x}_n &= \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n \\ &\approx \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{F}_x (\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{q}_n\end{aligned}$$

Note that $\mathbf{F}_x = \mathbf{F}_x(\hat{\mathbf{x}}_{n-1|n-1})$.

- Predicted mean:

$$\begin{aligned}\hat{\mathbf{x}}_{n|n-1} &= \mathbb{E}\{\mathbf{x}_n \mid \mathbf{y}_{1:n-1}\} \\ &\approx \mathbb{E}\{\mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{F}_x (\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{q}_n \mid \mathbf{y}_{1:n-1}\} \\ &= \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{F}_x \mathbb{E}\{\mathbf{x}_{n-1} \mid \mathbf{y}_{1:n-1}\} - \mathbf{F}_x \hat{\mathbf{x}}_{n-1|n-1} \\ &= \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{F}_x \hat{\mathbf{x}}_{n-1|n-1} - \mathbf{F}_x \hat{\mathbf{x}}_{n-1|n-1} \\ &= \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1})\end{aligned}$$

Linearized Model: Prediction (2/2)

- State estimate from t_{n-1} : $\hat{\mathbf{x}}_{n-1|n-1}$, $\mathbf{P}_{n-1|n-1}$
- Linearization around $\hat{\mathbf{x}}_{n-1|n-1}$ (dynamic model):

$$\mathbf{x}_n \approx \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{F}_x (\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{q}_n,$$

- Covariance:

$$\begin{aligned}\mathbf{P}_{n|n-1} &= E\{(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^T \mid \mathbf{y}_{1:n-1}\} \\ &\approx E\{[\mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{F}_x(\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{q}_n - \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1})] \\ &\quad \times [\dots]^T \mid \mathbf{y}_{1:n-1}\} \\ &= E\{[\mathbf{F}_x(\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{q}_n][\dots]^T \mid \mathbf{y}_{1:n-1}\} \\ &= \mathbf{F}_x E\{(\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1})(\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1})^T \mid \mathbf{y}_{1:n-1}\} \mathbf{F}_x^T \\ &\quad + E\{\mathbf{q}_n \mathbf{q}_n^T \mid \mathbf{y}_{1:n-1}\} \\ &= \mathbf{F}_x \mathbf{P}_{n-1|n-1} \mathbf{F}_x^T + \mathbf{Q}_n\end{aligned}$$

Linearized Model: Measurement Update (1/3)

- Prediction from t_{n-1} to t_n : $\hat{\mathbf{x}}_{n|n-1}$, $\mathbf{P}_{n|n-1}$
- Linearization around $\hat{\mathbf{x}}_{n|n-1}$:

$$\begin{aligned}\mathbf{y}_n &= \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n \\ &\approx \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}) + \mathbf{G}_x(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) + \mathbf{r}_n\end{aligned}$$

- Regularized linear least squares:

$$\begin{aligned}J_{\text{ReLS}}(\mathbf{x}_n) &= (\mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}) - \mathbf{G}_x(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}))^\top \mathbf{R}_n^{-1} \\ &\quad \times (\mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}) - \mathbf{G}_x(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})) \\ &\quad + (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^\top \mathbf{P}_{n|n-1}^{-1} (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) \\ \hat{\mathbf{x}}_{n|n} &= \underset{\mathbf{x}_n}{\operatorname{argmin}} J_{\text{ReLS}}(\mathbf{x}_n)\end{aligned}$$

Linearized Model: Measurement Update (2/3)

- Regularized linear least squares:

$$\begin{aligned} J_{\text{ReLS}}(\mathbf{x}_n) = & (\mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}) - \mathbf{G}_x(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}))^\top \mathbf{R}_n^{-1} \\ & \times (\mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}) - \mathbf{G}_x(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})) \\ & + (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^\top \mathbf{P}_{n|n-1}^{-1} (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) \end{aligned}$$

- Change of variables: $\mathbf{z}_n = \mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}) + \mathbf{G}_x \hat{\mathbf{x}}_{n|n-1}$:

$$\begin{aligned} J_{\text{ReLS}}(\mathbf{x}_n) = & (\mathbf{z}_n - \mathbf{G}_x \mathbf{x}_n)^\top \mathbf{R}_n^{-1} (\mathbf{z}_n - \mathbf{G}_x \mathbf{x}_n) \\ & + (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^\top \mathbf{P}_{n|n-1}^{-1} (\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) \end{aligned}$$

- Solution (see Chapters 2.4, 5.2):

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{z}_n - \mathbf{G}_x \hat{\mathbf{x}}_{n|n-1})$$

$$\mathbf{K}_n = \mathbf{P}_{n|n-1} \mathbf{G}_x^\top (\mathbf{G}_x \mathbf{P}_{n|n-1} \mathbf{G}_x^\top + \mathbf{R}_n)^{-1}$$

$$\mathbf{P}_{n|n} \approx \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_x \mathbf{P}_{n|n-1} \mathbf{G}_x^\top + \mathbf{R}_n) \mathbf{K}_n^\top$$

Linearized Model: Measurement Update (3/3)

- Measurement update:

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n(\mathbf{z}_n - \mathbf{G}_x \hat{\mathbf{x}}_{n|n-1})$$

- Substitution of $\mathbf{z}_n = \mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}) + \mathbf{G}_x \hat{\mathbf{x}}_{n|n-1}$:

$$\begin{aligned}\hat{\mathbf{x}}_{n|n} &= \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n(\mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}) + \mathbf{G}_x \hat{\mathbf{x}}_{n|n-1} - \mathbf{G}_x \hat{\mathbf{x}}_{n|n-1}) \\ &= \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n(\mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}))\end{aligned}$$

Linearized Model: Summary

- Model approximation:

$$\mathbf{x}_n \approx \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{F}_x(\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{q}_n$$

$$\mathbf{y}_n \approx \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}) + \mathbf{G}_x(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) + \mathbf{r}_n$$

- Prediction:

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}),$$

$$\mathbf{P}_{n|n-1} = \mathbf{F}_x \mathbf{P}_{n-1|n-1} \mathbf{F}_x^T + \mathbf{Q}_n,$$

- Measurement update:

$$\mathbf{K}_n = \mathbf{P}_{n|n-1} \mathbf{G}_x^T (\mathbf{G}_x \mathbf{P}_{n|n-1} \mathbf{G}_x^T + \mathbf{R}_n)^{-1},$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1})),$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_x \mathbf{P}_{n|n-1} \mathbf{G}_x^T + \mathbf{R}_n) \mathbf{K}_n^T.$$

Extended Kalman Filter

Algorithm 1 Extended Kalman Filter

1: Initialize $\hat{\mathbf{x}}_{0|0} = \mathbf{m}_0$, $\mathbf{P}_{0|0} = \mathbf{P}_0$

2: **for** $n = 1, 2, \dots$ **do**

3: Prediction (time update):

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1})$$

$$\mathbf{P}_{n|n-1} = \mathbf{F}_x \mathbf{P}_{n-1|n-1} \mathbf{F}_x^T + \mathbf{Q}_n$$

4: Measurement update:

$$\mathbf{K}_n = \mathbf{P}_{n|n-1} \mathbf{G}_x^T (\mathbf{G}_x \mathbf{P}_{n|n-1} \mathbf{G}_x + \mathbf{R}_n)^{-1}$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}))$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_x \mathbf{P}_{n|n-1} \mathbf{G}_x + \mathbf{R}_n) \mathbf{K}_n^T$$

5: **end for**

Example: Object Tracking (1/3)

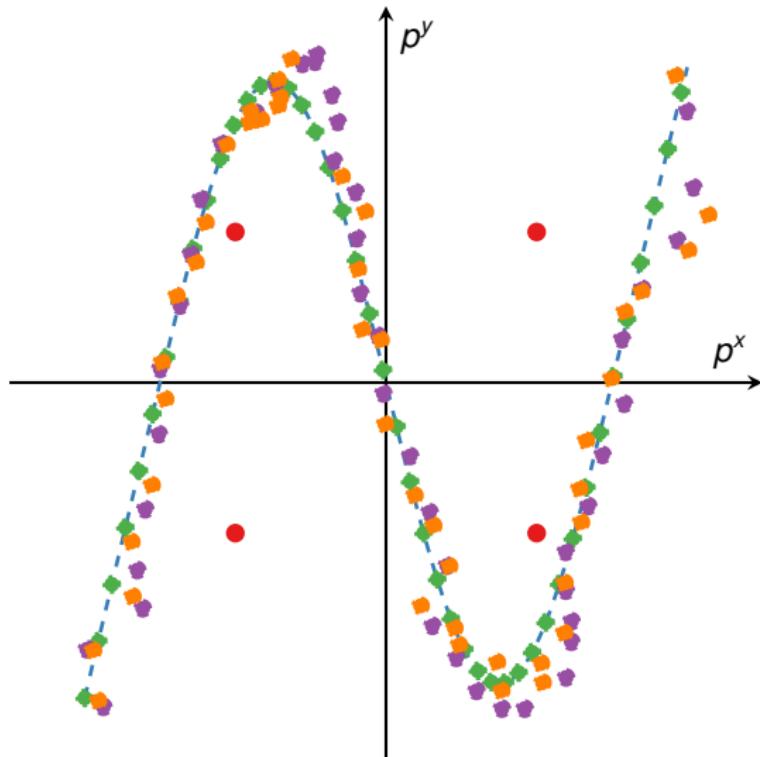
- Quasi-constant turn model:

$$\begin{bmatrix} \dot{p}^x(t) \\ \dot{p}^y(t) \\ \dot{v}(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} v(t) \cos(\varphi(t)) \\ v(t) \sin(\varphi(t)) \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{w}(t)$$

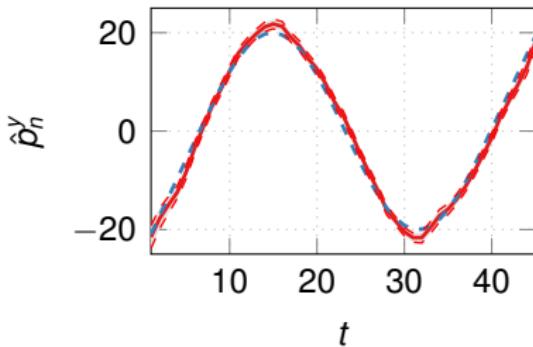
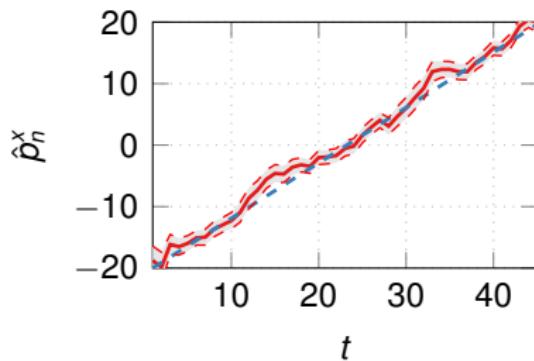
- We can use Euler–Maruyama to discretize this.
- Range (distance) measurements:

$$\mathbf{y}_n = \begin{bmatrix} |\mathbf{p}_n - \mathbf{p}_1^s| \\ |\mathbf{p}_n - \mathbf{p}_2^s| \\ \vdots \\ |\mathbf{p}_n - \mathbf{p}_K^s| \end{bmatrix} + \mathbf{r}_n$$

Example: Object Tracking (2/3)



Example: Object Tracking (3/3)

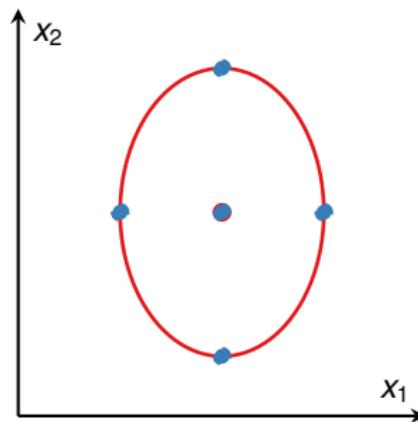


Position RMSE: 3.83 m

Nonlinear Transformations of Random Variables (1/3)

- Given: Random variable \mathbf{x} with mean \mathbf{m} and covariance \mathbf{P}
- Choose points \mathbf{x}^j and weights w_m^j, w_P^j such that:

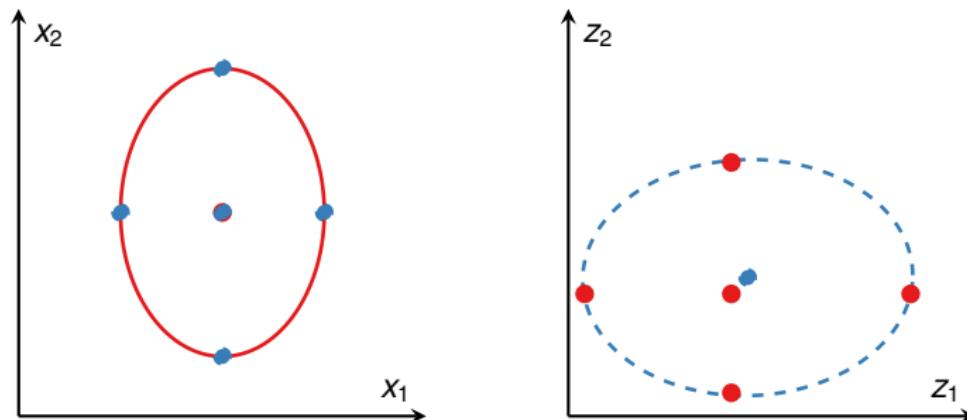
$$\mathbf{m} = \sum_{j=0}^{J-1} w_m^j \mathbf{x}^j, \mathbf{P} = \sum_{j=0}^{J-1} w_P^j (\mathbf{x}^j - \mathbf{m})(\mathbf{x}^j - \mathbf{m})^T,$$



Nonlinear Transformations of Random Variables (2/3)

- Given: Points \mathbf{x}^j and weights w_m^j, w_P^j
- Nonlinear transformation: $\mathbf{z} = \mathbf{h}(\mathbf{x})$
- Transformed points:

$$\mathbf{z}^j = \mathbf{h}(\mathbf{x}^j)$$



Nonlinear Transformations of Random Variables

(3/3)

- Given: Points \mathbf{x}^j and weights w_m^j, w_P^j
- Nonlinear transformation: $\mathbf{z} = \mathbf{h}(\mathbf{x})$
- Transformed points:

$$\mathbf{z}^j = \mathbf{h}(\mathbf{x}^j)$$

- Moments of the transformed variable

$$E\{\mathbf{z}\} \approx \sum_{j=1}^J w_m^j \mathbf{z}^j$$

$$\text{Cov}\{\mathbf{z}\} \approx \sum_{j=1}^J w_P^j (\mathbf{z}^j - E\{\mathbf{z}\})(\mathbf{z}^j - E\{\mathbf{z}\})^T$$

$$\text{Cov}\{\mathbf{x}, \mathbf{z}\} \approx \sum_{j=1}^J w_P^j (\mathbf{x}^j - \mathbf{m})(\mathbf{z}^j - E\{\mathbf{z}\})^T$$

Unscented Transform

- **Unscented Transform:** One way of choosing \mathbf{x}^j , w_m^j and w_P^j , uses $2L + 1$ points
- Location of the sigma-points:

$$\mathbf{x}^0 = \mathbf{m}$$

$$\mathbf{x}^j = \mathbf{m} + \sqrt{L + \lambda} [\sqrt{\mathbf{P}}]_j, \quad j = 1, \dots, L$$

$$\mathbf{x}^j = \mathbf{m} - \sqrt{L + \lambda} [\sqrt{\mathbf{P}}]_{(j-L)}, \quad j = L + 1, \dots, 2L$$

- Weights of the sigma-points:

$$w_m^0 = \frac{\lambda}{L + \lambda}$$

$$w_P^0 = \frac{\lambda}{L + \lambda} + (1 - \alpha^2 + \beta)$$

$$w_m^j = w_P^j = \frac{1}{2(L + \lambda)}, \quad j = 1, \dots, 2L$$

Unscented Transform: Prediction (1/2)

- Dynamic model:

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n$$

- Sigma-points with $\mathbf{m} = \hat{\mathbf{x}}_{n-1|n-1}$, $\mathbf{P} = \mathbf{P}_{n-1|n-1}$:

$$\mathbf{x}_{n-1}^0 = \hat{\mathbf{x}}_{n-1|n-1}$$

$$\mathbf{x}_{n-1}^j = \hat{\mathbf{x}}_{n-1|n-1} + \sqrt{L + \lambda} \left[\sqrt{\mathbf{P}_{n-1|n-1}} \right]_j, \quad j = 1, \dots, L$$

$$\mathbf{x}_{n-1}^j = \hat{\mathbf{x}}_{n-1|n-1} - \sqrt{L + \lambda} \left[\sqrt{\mathbf{P}_{n-1|n-1}} \right]_{(j-L)}, \quad j = L + 1, \dots, 2L$$

Unscented Transform: Prediction (2/2)

- Dynamic model:

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n$$

- Transformed points:

$$\mathbf{x}_n^j = \mathbf{f}(\mathbf{x}_{n-1}^j), \quad j = 0, \dots, 2L$$

- Moments of the prediction:

$$\hat{\mathbf{x}}_{n|n-1} = \sum_{j=0}^{2L} w_m^j \mathbf{x}_n^j$$

$$\mathbf{P}_{n|n-1} = \sum_{j=0}^{2L} w_c^j (\mathbf{x}_n^j - \hat{\mathbf{x}}_{n|n-1})(\mathbf{x}_n^j - \hat{\mathbf{x}}_{n|n-1})^\top + \mathbf{Q}_n$$

Unscented Transform: Measurement Update (1/2)

- Measurement model:

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n$$

- Recall: Alternative form of measurement update:

$$\mathbf{K}_n = \text{Cov}\{\mathbf{x}_n, \mathbf{y}_n | \mathbf{y}_{1:n-1}\} \text{Cov}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}^{-1},$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbb{E}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}),$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n \text{Cov}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\} \mathbf{K}_n^T.$$

- We can calculate $\mathbb{E}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}$, $\text{Cov}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}$, and $\text{Cov}\{\mathbf{x}_n, \mathbf{y}_n | \mathbf{y}_{1:n-1}\}$ using the unscented transform
- Sigma-points based on $\hat{\mathbf{x}}_{n|n-1}$, $\mathbf{P}_{n|n-1}$:

$$\mathbf{x}_n^0 = \hat{\mathbf{x}}_{n|n-1}$$

$$\mathbf{x}_n^j = \hat{\mathbf{x}}_{n|n-1} + \sqrt{L + \lambda} \left[\sqrt{\mathbf{P}_{n|n-1}} \right]_j, \quad j = 1, \dots, L$$

$$\mathbf{x}_n^j = \hat{\mathbf{x}}_{n|n-1} - \sqrt{L + \lambda} \left[\sqrt{\mathbf{P}_{n|n-1}} \right]_{(j-L)}, \quad j = L + 1, \dots, 2L$$

Unscented Transform: Measurement Update (2/2)

- Measurement model:

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n$$

- Transformed sigma-points:

$$\mathbf{y}_n^j = \mathbf{g}(\mathbf{x}_n^j), \quad j = 0, \dots, 2L$$

- Moments of the predicted \mathbf{y}_n :

$$\mathbb{E}\{\mathbf{y}_n \mid \mathbf{y}_{1:n-1}\} = \sum_{j=0}^{2L} w_m^j \mathbf{y}_n^j$$

$$\begin{aligned} \text{Cov}\{\mathbf{y}_n \mid \mathbf{y}_{1:n-1}\} &= \sum_{j=0}^{2L} w_P^j (\mathbf{y}_n^j - \mathbb{E}\{\mathbf{y}_n \mid \mathbf{y}_{1:n-1}\}) \\ &\quad \times (\mathbf{y}_n^j - \mathbb{E}\{\mathbf{y}_n \mid \mathbf{y}_{1:n-1}\})^\top + \mathbf{R}_n \end{aligned}$$

$$\text{Cov}\{\mathbf{x}_n, \mathbf{y}_n \mid \mathbf{y}_{1:n-1}\} = \sum_{j=0}^{2L} w_P^j (\mathbf{x}_n^j - \hat{\mathbf{x}}_{n|n-1})(\mathbf{y}_n^j - \mathbb{E}\{\mathbf{y}_n \mid \mathbf{y}_{1:n-1}\})^\top$$

Unscented Kalman Filter

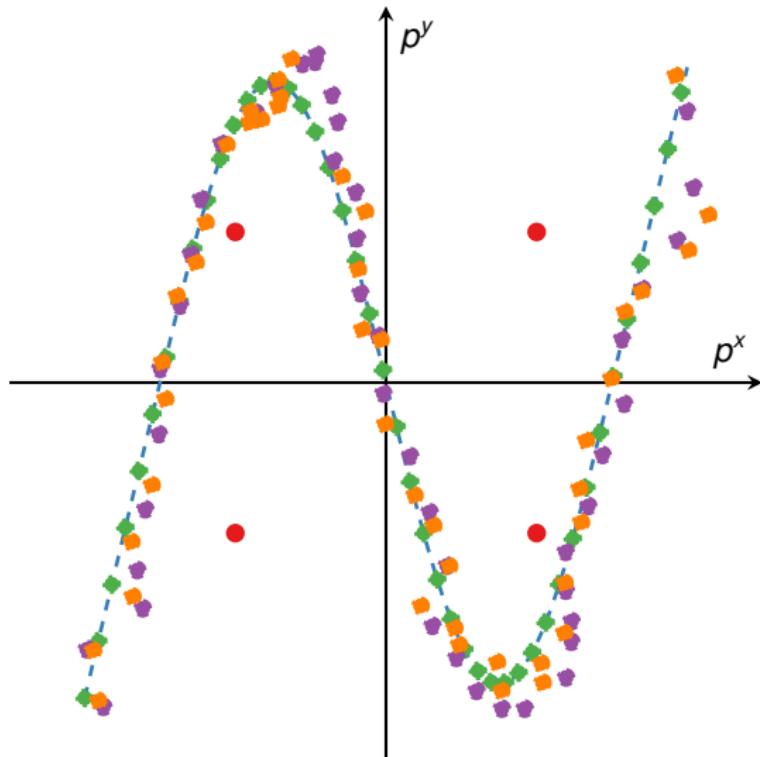
- Prediction:
 - Calculate the sigma-points using $\hat{\mathbf{x}}_{n-1|n-1}$ and $\mathbf{P}_{n-1|n-1}$
 - Propagate the sigma-points $\mathbf{x}_n^j = \mathbf{f}(\mathbf{x}_{n-1}^j)$
 - Calculate the mean and covariance $\hat{\mathbf{x}}_{n|n-1}$, $\mathbf{P}_{n|n-1}$
- Measurement update:
 - Calculate the sigma-points using $\hat{\mathbf{x}}_{n|n-1}$ and $\mathbf{P}_{n|n-1}$
 - Propagate the sigma-points $\mathbf{y}_n^j = \mathbf{g}(\mathbf{x}_n^j)$
 - Calculate the mean and covariance $E\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}$,
 $\text{Cov}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}$, $\text{Cov}\{\mathbf{x}_n, \mathbf{y}_n | \mathbf{y}_{1:n-1}\}$
 - Perform the Kalman filter measurement update:

$$\mathbf{K}_n = \text{Cov}\{\mathbf{x}_n, \mathbf{y}_n | \mathbf{y}_{1:n-1}\} \text{Cov}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}^{-1},$$

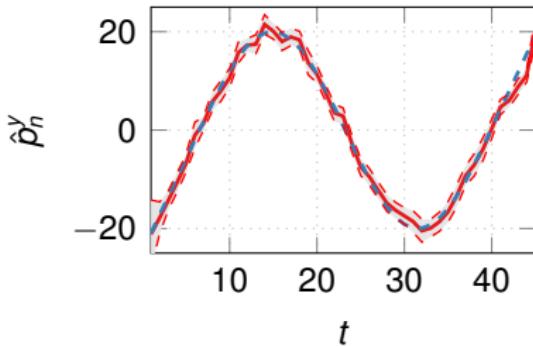
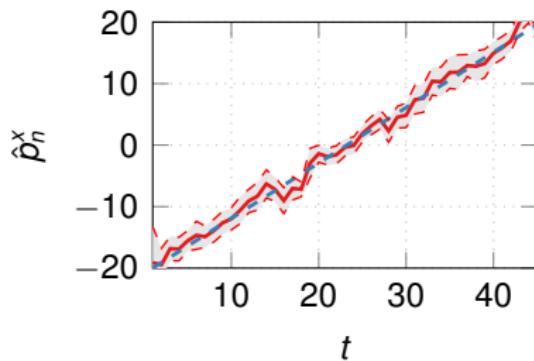
$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - E\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}),$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n \text{Cov}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\} \mathbf{K}_n^T.$$

Example: Object Tracking (1/2)



Example: Object Tracking (2/2)



Position RMSE: 1.45 m

Unscented Transform: Choice of Parameters

- The parameter λ is actually:

$$\lambda = \alpha^2(L + \kappa) - L$$

- α , β , and κ are tuning parameters
- κ is usually set to 0
- α controls the spread of the sigma-points:

$$\sqrt{L + \lambda} = \sqrt{L + \alpha^2(L + \kappa) - L} = \alpha\sqrt{L}.$$

- Suggestions vary, e.g., $\alpha = 1 \times 10^{-3}$
- β only affects the covariance weight, a good starting point is $\beta = 2$

Summary

- Nonlinear state-space models require approximative solutions
- The extended Kalman filter uses a linearization of the dynamic and measurement models
- The unscented Kalman filter uses a set of deterministic sigma-points (samples) to calculate the means and covariances



Aalto University
School of Electrical
Engineering

ELEC-E8740 — Bootstrap Particle Filtering

Simo Särkkä

Aalto University

November 27, 2020

Contents

- 1 Intended Learning Outcomes and Recap
- 2 Idea of Particle Filter
- 3 Resampling and Particle Filter Algorithm
- 4 Summary

Intended Learning Outcomes

After this lecture, you will be able to:

- describe the basic idea of particle filtering,
- explain the three steps in particle filtering: simulation, weighting, resampling,
- identify the differences between Kalman filtering and particle filtering.

Recap: Extended Kalman Filter

- Model approximation:

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n \approx \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{F}_x(\mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{q}_n$$

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n \approx \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}) + \mathbf{G}_x(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1}) + \mathbf{r}_n$$

- Prediction:

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1}),$$

$$\mathbf{P}_{n|n-1} = \mathbf{F}_x \mathbf{P}_{n-1|n-1} \mathbf{F}_x^T + \mathbf{Q}_n,$$

- Measurement update:

$$\mathbf{K}_n = \mathbf{P}_{n|n-1} \mathbf{G}_x^T (\mathbf{G}_x \mathbf{P}_{n|n-1} \mathbf{G}_x^T + \mathbf{R}_n)^{-1},$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1})),$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n (\mathbf{G}_x \mathbf{P}_{n|n-1} \mathbf{G}_x^T + \mathbf{R}_n) \mathbf{K}_n^T.$$

Recap: Unscented Kalman Filter

- Uses a nonlinear transformation of deterministic sampling points
- Prediction:
 - Calculate the sigma-points using $\hat{\mathbf{x}}_{n-1|n-1}$ and $\mathbf{P}_{n-1|n-1}$
 - Propagate the sigma-points $\mathbf{x}_n^j = \mathbf{f}(\mathbf{x}_{n-1}^j)$
 - Calculate the mean and covariance $\hat{\mathbf{x}}_{n|n-1}$, $\mathbf{P}_{n|n-1}$
- Measurement update:
 - Calculate the sigma-points using $\hat{\mathbf{x}}_{n|n-1}$ and $\mathbf{P}_{n|n-1}$
 - Propagate the sigma-points $\mathbf{y}_n^j = \mathbf{g}(\mathbf{x}_n^j)$
 - Calculate the mean and covariance $E\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}$,
 $\text{Cov}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}$, $\text{Cov}\{\mathbf{x}_n, \mathbf{y}_n | \mathbf{y}_{1:n-1}\}$
 - Perform the Kalman filter measurement update:

$$\mathbf{K}_n = \text{Cov}\{\mathbf{x}_n, \mathbf{y}_n | \mathbf{y}_{1:n-1}\} \text{Cov}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}^{-1},$$

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n (\mathbf{y}_n - E\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\}),$$

$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n \text{Cov}\{\mathbf{y}_n | \mathbf{y}_{1:n-1}\} \mathbf{K}_n^T.$$

Discrete-Time Nonlinear State-Space Model

- Discrete-time nonlinear state-space model:

$$\begin{aligned}\mathbf{x}_n &= \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n \\ \mathbf{y}_n &= \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n\end{aligned}$$

- Process noise: $\mathbf{q}_n \sim p(\mathbf{q}_n)$
- Measurement noise: $\mathbf{r}_n \sim p(\mathbf{r}_n)$
- Initial state: $\mathbf{x}_0 \sim p(\mathbf{x}_0)$

This is a stochastic process, each realization of the state sequence $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ is different

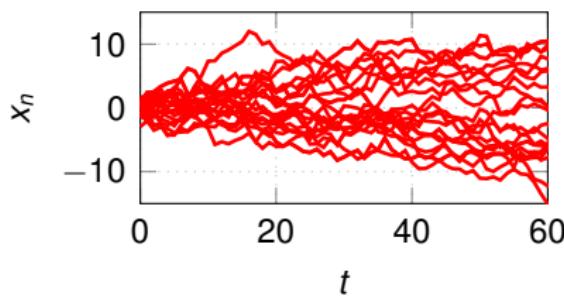
Example: Random Walk Process (1/2)

- Dynamic model:

$$x_n = x_{n-1} + q_n$$

$$x_0 \sim \mathcal{N}(0, 1)$$

$$q_n \sim \mathcal{N}(0, 1)$$

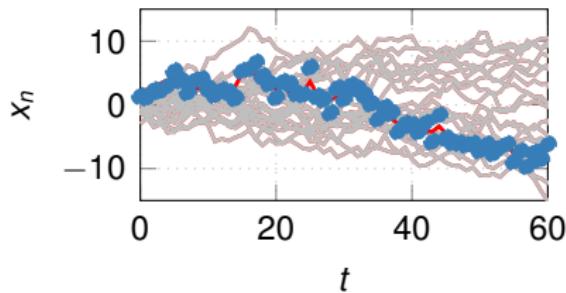


Example: Random Walk Process (2/2)

- Only one realization of the process is observed
- Measurement model:

$$y_n = x_n + r_n$$

$$r_n \sim \mathcal{N}(0, 1)$$



Particle Filtering: Idea

Prediction

- Given: Simulated states \mathbf{x}_{n-1}^j ($j = 1, \dots, J$)
- Simulate from t_{n-1} to t_n to obtain \mathbf{x}_n^j ($j = 1, \dots, J$)

Measurement Update

- Evaluate how well \mathbf{x}_n^j explains \mathbf{y}_n ($j = 1, \dots, J$)
- Assign a weight w_n^j to \mathbf{x}_n^j ($j = 1, \dots, J$)

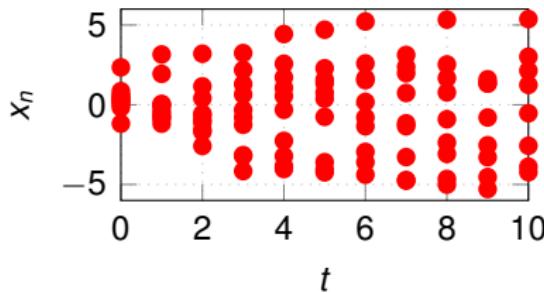
Prediction: Simulation

- Intuitive way: Use the dynamic model to simulate one time step

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{q}_n$$

- Two step procedure:

- 1 Sample $\mathbf{q}_n^j \sim p(\mathbf{q}_n)$,
- 2 Calculate $\mathbf{x}_n^j = \mathbf{f}(\mathbf{x}_{n-1}^j) + \mathbf{q}_n^j$.



Measurement Update: Importance Weights

- Weights w_n^j indicate the relevance of each sample
- Importance weights:
 - High weight w_n^j : Explains \mathbf{y}_n well
 - Low weight w_n^j : Explains \mathbf{y}_n poorly
 - Should sum to one:

$$\sum_{j=1}^J w_n^j = 1,$$

- Cost function gives low values for good estimates of \mathbf{x}_n

Measurement Update: Likelihood (1/2)

- Measurement model:

$$\begin{aligned}\mathbf{y}_n &= \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n \\ \mathbf{r}_n &\sim p(\mathbf{r}_n)\end{aligned}$$

- \mathbf{r}_n is a random variable $\Rightarrow \mathbf{y}_n$ is a random variable too
- \mathbf{y}_n must have a probability density function (pdf)
- Given \mathbf{x}_n , the pdf for \mathbf{y}_n is the same as for \mathbf{r}_n but shifted by $\mathbf{g}(\mathbf{x}_n)$
- The pdf for \mathbf{y}_n given \mathbf{x}_n is called the likelihood

$$\mathbf{y}_n \sim p(\mathbf{y}_n \mid \mathbf{x}_n)$$

Measurement Update: Likelihood (2/2)

- The likelihood is a suitable measure for the importance weights w_n^j
- The non-normalized weights are then:

$$\tilde{w}_n^j = p(\mathbf{y}_n \mid \mathbf{x}_n^j).$$

- Normalization:

$$w_n^j = \frac{\tilde{w}_n^j}{\sum_{i=1}^J \tilde{w}_n^i}.$$

Example: Gaussian Likelihood (1/2)

- Measurement model:

$$\mathbf{y}_n = \mathbf{g}(\mathbf{x}_n) + \mathbf{r}_n$$

- The measurement noise is often (assumed) Gaussian:

$$p(\mathbf{r}_n) = \mathcal{N}(\mathbf{r}_n; \mathbf{0}, \mathbf{R}_n)$$

- Then, the likelihood is Gaussian too:

$$p(\mathbf{y}_n \mid \mathbf{x}_n) = \mathcal{N}(\mathbf{y}_n; \mathbf{g}(\mathbf{x}_n), \mathbf{R}_n).$$

Example: Gaussian Likelihood (2/2)

- Example: Scalar case with

$$y_n = \mathbf{g}(x_n) + r_n$$

$$r_n \sim \mathcal{N}(0, \sigma_r^2)$$

Point Estimates

- Moments of the state can be calculated using weighted sums of the weighted samples
- Mean:

$$\hat{\mathbf{x}}_{n|n} = \sum_{j=1}^J w_n^j \mathbf{x}_n^j$$

- Covariance:

$$\mathbf{P}_{n|n} = \sum_{j=1}^J w_n^j (\mathbf{x}_n^j - \hat{\mathbf{x}}_{n|n})(\mathbf{x}_n^j - \hat{\mathbf{x}}_{n|n})^\top.$$

Summary: Sequential Sampling and Weighing

- Initialization: Sample J particles:

$$\mathbf{x}_0^j \sim p(\mathbf{x}_0)$$

- Prediction: Sample \mathbf{q}_n^j and propagate particles:

$$\mathbf{q}_n^j \sim p(\mathbf{q}_n)$$

$$\mathbf{x}_n^j = \mathbf{f}(\mathbf{x}_n^j) + \mathbf{q}_n^j$$

- Measurement update: Calculate and normalize the particle weights:

$$\tilde{w}_n^j = p(\mathbf{y}_n \mid \mathbf{x}_n^j)$$

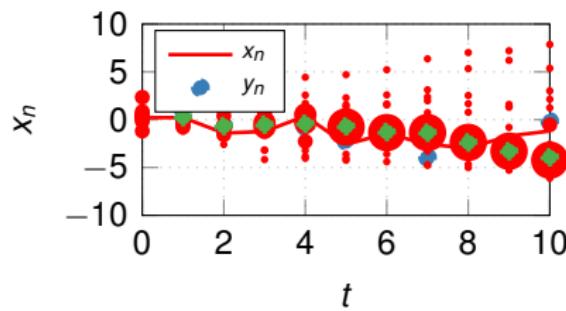
$$w_n^j = \frac{w_n^j}{\sum_{i=1}^J \tilde{w}_n^i}$$

Example: Random Walk Process

- State-space model:

$$x_n = x_{n-1} + q_n$$

$$y_n = x_n + r_n$$



Resampling (1/2)

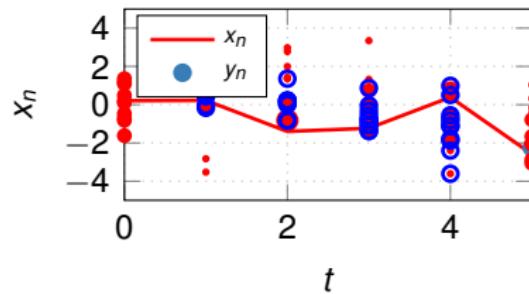
- Problem: The particles diverge after a few samples
- Resampling:
 - Remove samples with low weights
 - Replicate samples with high weights
 - Samples should be represented proportional to their weight:

$$\lfloor w_n^j \rfloor$$

- Equivalent interpretation

$$\Pr\{\tilde{\mathbf{x}}_n^i = \mathbf{x}_n^j\} = w_n^j,$$

Resampling (2/2)



Bootstrap Particle Filter

Algorithm 1 Bootstrap Particle Filter (Gaussian Noises)

- 1: Initialize: $\mathbf{x}_0^j \sim \mathcal{N}(\mathbf{m}_0, \mathbf{P}_0)$ ($j = 1, \dots, J$)
- 2: **for** $n = 1, 2, \dots$ **do**
- 3: **for** $j = 1, 2, \dots, J$ **do**
- 4: Sample: $\mathbf{q}_n^j \sim \mathcal{N}(0, \mathbf{Q})$
- 5: Propagate the state: $\mathbf{x}_n^j = \mathbf{f}(\mathbf{x}_{n-1}^j) + \mathbf{q}_n^j$
- 6: Calculate the weights: $\tilde{w}_n^j = \mathcal{N}(\mathbf{y}_n; \mathbf{g}(\mathbf{x}_n^j), \mathbf{R}_n)$
- 7: **end for**
- 8: Normalize the importance weights ($j = 1, \dots, J$)

$$w_n^j = \frac{\tilde{w}_n^j}{\sum_{i=1}^J \tilde{w}_n^i}$$

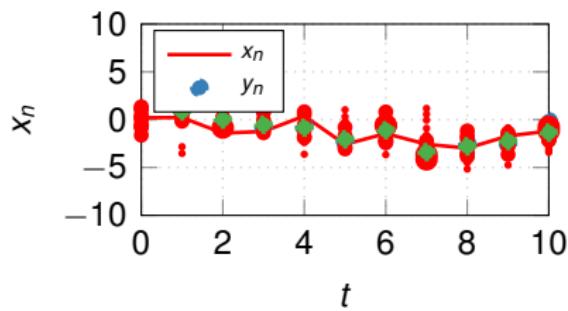
- 9: Calculate the mean $\hat{\mathbf{x}}_{n|n}$ and covariance $\mathbf{P}_{n|n}$
- 10: Resample such that $\Pr\{\tilde{\mathbf{x}}_n^i = \mathbf{x}_n^j\} = w_n^j$
- 11: **end for**

Example: Random Walk

- State-space model:

$$x_n = x_{n-1} + q_n$$

$$y_n = x_n + r_n$$



Example: Object Tracking (1/3)

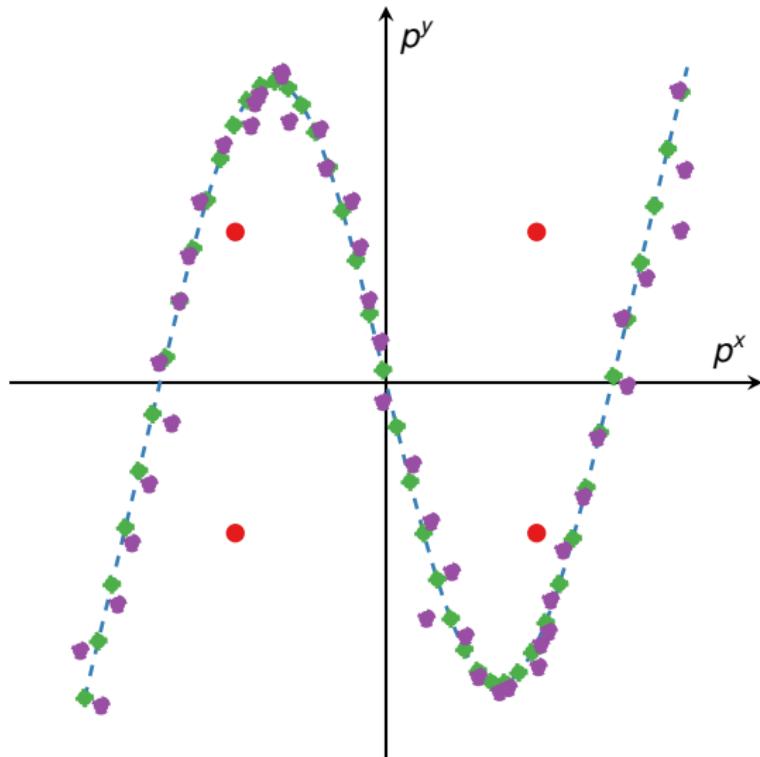
- Quasi-constant turn model:

$$\begin{bmatrix} \dot{p}^x(t) \\ \dot{p}^y(t) \\ \dot{v}(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} v(t) \cos(\varphi(t)) \\ v(t) \sin(\varphi(t)) \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{w}(t)$$

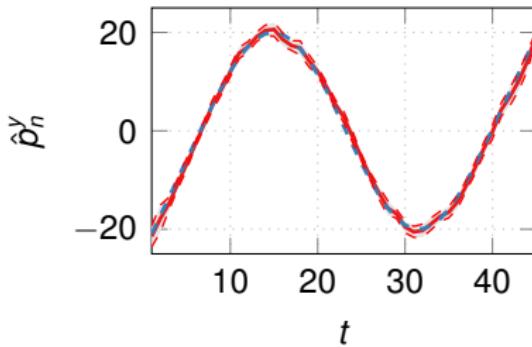
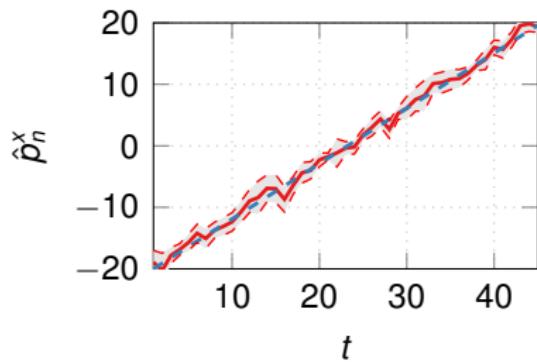
- Range (distance) measurements:

$$\mathbf{y}_n = \begin{bmatrix} |\mathbf{p}_n - \mathbf{p}_1^s| \\ |\mathbf{p}_n - \mathbf{p}_2^s| \\ \vdots \\ |\mathbf{p}_n - \mathbf{p}_K^s| \end{bmatrix} + \mathbf{r}_n$$

Example: Object Tracking (2/3)



Example: Object Tracking (3/3)



Summary

- The particle filter uses a set of random samples to estimate the state
- During prediction, the samples are simulated from t_{n-1} to t_n
- The **bootstrap particle filter** uses the dynamic model to simulate the samples
- The measurement update evaluates the likelihood to assign an **importance weight** to each sample
- Resampling is used to mitigate **particle degeneracy**
- Particle filtering is a **universal approach** equally applicable to linear and nonlinear system
- It can be shown that particle filters are asymptotically ($J \rightarrow \infty$) optimal in many cases