UKF notes

1. Compare with ekf

Ekf: uses Jacobean matrix to linear in non-linear function

ukf: takes representative points from Gaussian distribution. These points will be plugged into the non-linear equations

此外，EKF 将非线性状态方程和观测方程线性化（使用泰勒展开，且使用低阶忽略高阶【使用高阶会增加很多计算量】)，再使用 KF

缺点：对于非线性程度较大的模型（强非线性时，忽略高阶会带来较大的误差，会使滤波发散）

UKF 可以解决强非线性的问题，且省略了繁琐的雅可比矩阵计算。

- what problem does UKF solve

- if the process model is non-linear, that is, the prediction is defined by a nonlinear function.
- it will provides a distribution which is not normally distributed any more. (可以参考将高斯分布看作每个小点，对小点进行非线性操作)
- However, ukf keeps going as if the prediction was still normal.

-what we want to find is the normal distribution that represents the real predicted distribution as close as possible. (ukf aim)

-UKF Basics: Unscented Transformation

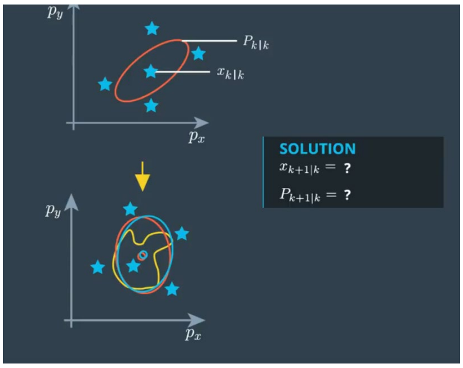
The Unscented Kalman filter finds the **mean vector** and covariance matrix using **sigma points.**

Problem: **difficult** to transform the **whole state distribution** through a nonlinear function, but it is very **easy** to transform **individual points of the state space** through the nonlinear function, and this is what sigma points are for.

-How does sigma points chosen?

They are chosen **around the mean state** and in a certain **relation to the standard deviation** (/sigma) of every state dimension. The serve as representatives of the whole distribution.

Process: choose sigma points -> push into nonlinear function -> find mean and covariance of results.



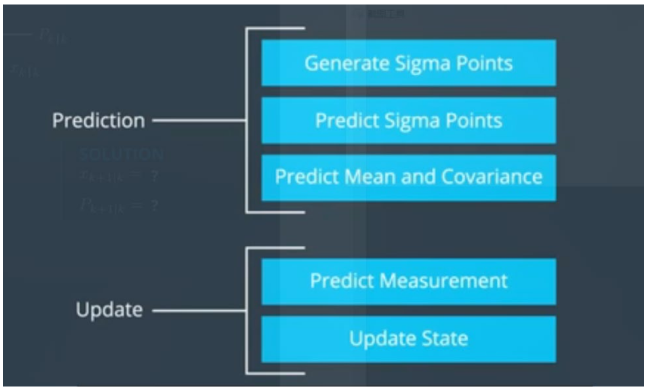
This **will not provide** the same mean and covariance as the real predicted distribution, but in many cases it gives a **useful approximation**.

Special case: linear case

You can apply this same technique in the linear case, you will find exact solution of the sigma points. → If the process model is linear, the sigma-point approach provides exactly the same solution as the standard common feature. But you will not use sigma point because they are more **expensive in terms of calculation time.**

-Process Chain Of UKF

Starting with a state vector x and a covariance matrix p, we will go all the way through prediction and the measurement step.



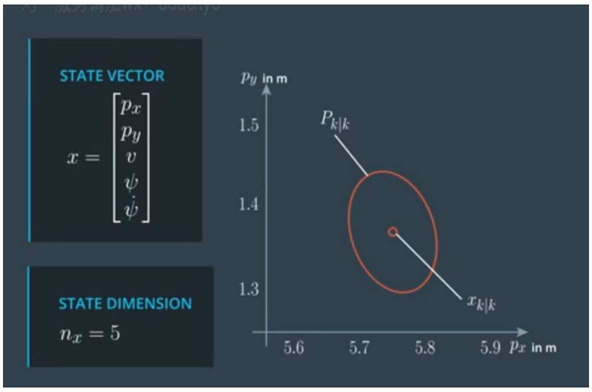
Start with Prediction step:

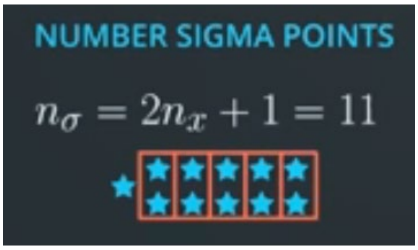
- We need to know a good way to choose sigma points
- We need to know how to predict the sigma points (i.e. Insert them into the process function)
- We need to know how to calculate the prediction(mean, and covariance from the predicted sigma points)

-Prediction 1:How to choose sigma point

At the beginning of the prediction step we have the posterior state **x_k_k** and the posterior covariance matrix **P_k_k** from the last iteration. They represent the distribution of our current state. For this distribution, we want to generate sigma points.

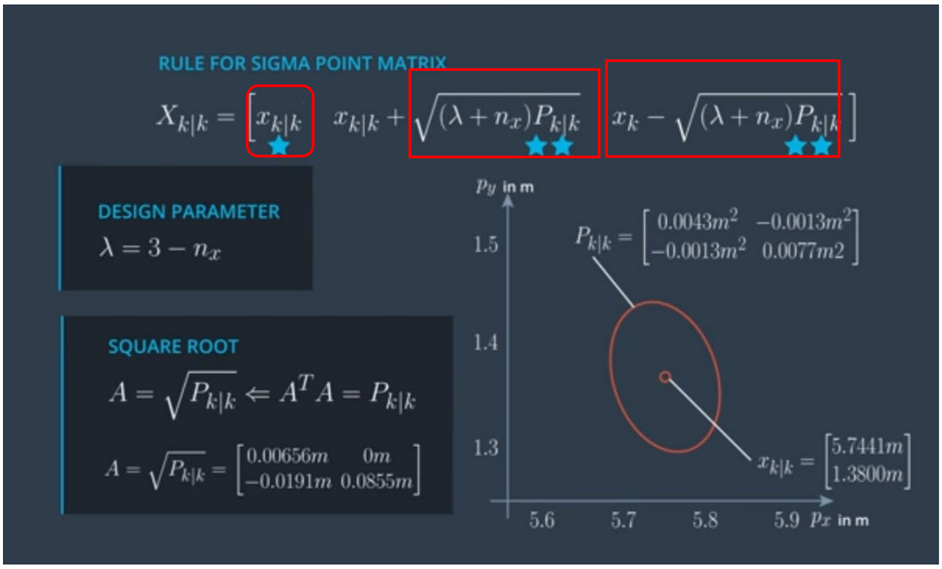
The **number** of sigma points depends on the **state dimension**. This is the state vector of the CTRV model so the dimension of our state is **nx = 5**. We will chose **2n_x +1** sigma points. The first point is the **mean** of the state. Then we have another two points per state dimension which will be spread in different directions.





-Simple example: state vector with two dimensions: [px, py]

State dimension = 2 → number of sigma points 2x2+1=5



解析：

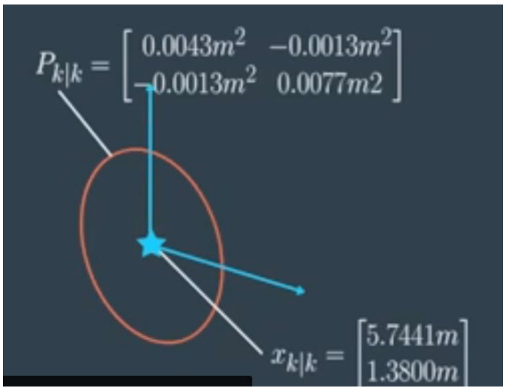
- **Lambda：design parameter**

  You can choose where in relation to the error ellipse you want to put your sigma points. Some people report good results with lambda = 3 – nx;
- **Square root of matrix 的定义： 如图**
- **对照 Rule for sigma point matrix：**

  1.第一项为 mean，告诉我们第一个 sigma point 在哪

  2.后两项对应 spread in different directions.

  Square root matrix 对应两个 vector， 第一个 vector 为矩阵的第一列，在如图所示的坐标系中定义了一个方向，第二个 vector 为矩阵的第二列，定义了另一个方向



3.lambda 与开方矩阵相乘：

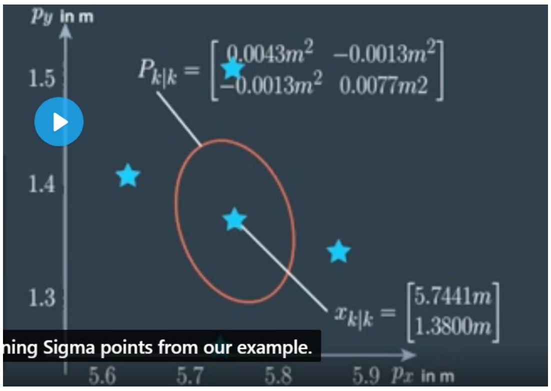
lambda larger -> sigma points move further away from mean state

lambda smaller ->sigma points move closer to mean state

$$X_{k|k} = \begin{bmatrix} 5.7441 & & 5.8577 & 5.7441 & x_{k|k} - \sqrt{(\lambda + n_x)P_{k|k}} \\ 1.3800 & & 1.3469 & 1.5281 & \end{bmatrix}$$

其实可以理解成 lambda 与两个 vector 分别相乘，所得结果还是两个 vector，再加到 mean 上，构成两个 sigma points，其实一个 vector 对应的是一个 state dimension(这里是 2)，往正方向运动也可以往反方向运动(后一项表示相反方向的运动)

这里生成的 sigma points 如下图所示：



Code：

https://github.com/mounchiliu/Udacity/tree/main/sensor-fusion/Unscented%20Kalman%20Filters/1.%20PREDICT%20--%20Sigma%20Point%20Generation