工程实践与科技创新 IV-E

Lecture 2: Kalman 滤波器

授课老师：张晗、王景川 *2022 年春季*

在上节课中，我们初步介绍了移动机器人建图、定位和导航、回顾了状态空间模型和一些概率知识、初步学习了离散时间隐 Markov 模型。在本次课中，我们将把上节课学到的知识，初步应用到机器人定位中——其主要数学框架是 Kalman 滤波。

1 天才的想法：时域滤波器

在《数字信号处理》课程中，我们已经接触过滤波器，主要用于滤除信号中的噪声。以下面离散时间的低通滤波器为例：

$$y_t = \alpha x_t + (1-\alpha) y_{t-1}$$

它的截止频率是 $f_c = \frac{\alpha}{(1-\alpha)2\pi\Delta t}$，其中 Δt 是采样周期。任何频率大于 f_c 的信号通过该滤波器都会得到极大的衰减，可以用于滤除高频噪声。

Figure 1: Rudolf Kalman

Kalman 滤波器则不同。它直接在**时域中**处理滤波问题，并且在线性系统下是**最小方差估计**。但天才的想法一开始往往不被人所接受。著名控制领域专家何毓琦先生回忆道：“当时几乎所有控制方面的著名工作都在 Fourier 与 Laplace 变换后的所谓频域里进行。Kalman 的方法则在动态系统上使用了时域微分方程模型。这在当时是非常离经叛道的，主流观点对此质疑很多。”1960 年，时任美国宇航局加利福尼亚州 Ames Research Center（ARC）动力分析处主任的斯坦尼 • 施密特（Stanley F. Schmidt）正在阿波罗-11 号登月计划中主持导航项目。当年，宇宙飞船从陀螺仪、加速度计和雷达等传感器上获取的测量数据中充满了不确定性误差和随机噪声，严重地威胁着高速飞向月球并降落其岩石表面的宇宙飞船及宇航员的安全。因此他们必须从测量数据中把噪声滤掉，以便对飞船所处位置和运动速度作出非常精确的估算。经过多方认证和周密思考之后，施密特决定在阿波罗-11 号登月计划中采用 Kalman 滤波器，并成功让阿波罗-11 号在地球和月球之间飞了一个来回。

2 最小（协）方差估计与后验概率传递

前文中提到，Kalman 滤波器是对信号的最小方差估计，其实现手段是通过迭代的方式计算隐 Markov 模型中隐状态的后验分布的数学期望。下面我们将简单介绍为什么后验分布的数学期望是方差最小的估计。

假设我们试图通过观测值 $y \in \mathbb{R}^m$，对一个随机向量 $x \in \mathbb{R}^n$ 可能的具体实现值进行估计（我们把该估计值记为 $\hat{x} \in \mathbb{R}^n$）。事实上，该估计值 $\hat{x}$ 应当是观测值 y 的一个函数，记为 $\hat{x} = g(y)$。我们的任务其实是构造某个函数 $g(y)$，使得估计值 $\hat{x}$ 满足某个最优指标（也即让某个指标 $\mathscr{L}(\hat{x}) = \mathscr{L}(g(y))$

达到最小值)。如果我们要求（协）方差最小，我们可把（协）方差写作

$$\begin{aligned}
\mathscr{L}(g(y)) &= \mathbb{E}\left[(x-g(y))(x-g(y))^T \mid y\right] \\
&= \mathbb{E}\left[xx^T - g(y)x^T - xg^T(y) + g(y)g^T(y) \mid y\right] \\
&= \mathbb{E}\left[xx^T \mid y\right] - g(y)\mathbb{E}[x \mid y]^T - \mathbb{E}[x \mid y]g^T(y) + g(y)g^T(y) \\
&= \mathbb{E}[xx^T \mid y] - \underbrace{\mathbb{E}\left[x \mid y\right]\mathbb{E}\left[x \mid y\right]^T}_{=\mathbb{E}[\mathbb{E}[x|y]x^T|y]} - \underbrace{\mathbb{E}\left[x \mid y\right]\mathbb{E}\left[x \mid y\right]^T}_{=\mathbb{E}[x\mathbb{E}[x|y]^T|y]} + \mathbb{E}\left[x \mid y\right]\mathbb{E}\left[x \mid y\right]^T + \mathbb{E}\left[x \mid y\right]\mathbb{E}\left[x \mid y\right]^T \\
&\quad - g(y)\mathbb{E}[x \mid y]^T - \mathbb{E}[x \mid y]g^T(y) + g(y)g^T(y) \\
&= \underbrace{\mathbb{E}\left[(x-\mathbb{E}[x \mid y])(x-\mathbb{E}[x \mid y])^T\right]}_{=\mathscr{L}(\mathbb{E}[x|y])} + \underbrace{(\mathbb{E}[x \mid y]-g(y))(\mathbb{E}[x \mid y]-g(y))^T}_{\text{正定矩阵}} \\
&\succeq \mathscr{L}(\mathbb{E}[x \mid y]).
\end{aligned}$$

从上式中可以看出，如果我们取 $g(y)=\mathbb{E}[x \mid y]$，（协）方差 $\mathbb{L}(g(y))$ 就可以取到最小值，所以后验概率分布的数学期望 $\mathbb{E}[x \mid y]$ 是最小（协）方差估计。

类似地，如果我们在 t 时刻，通过所有的历史观测信息 $y_{1:t}$ 对隐 Markov 模型

$$x_{t+1} = f(x_t) + w_t, \quad y_t = h(x_t) + v_t$$

的隐状态 x_t 的实现值进行估计，其最小（协方差）估计应为 $\mathbb{E}[x_t \mid y_{1:t}]$（其中 v_t 和 w_t 为白噪声，即不同时刻下的噪声互相独立)。**如果该隐 Markov 模型表示的是移动机器人运动学模型及其传感器模型，则对其状态（位姿）的估计（定位）就是其位姿在给定所有历史传感器数据下的数学期望值**。

为了计算最小（协）方差估计 $\mathbb{E}[x_t \mid y_{1:t}]$，我们需要找出该后验概率的概率密度函数 $p(x_t|y_{1:t})$。另一方面，我们希望**在线估计**，即当收到一个新的观测 y_{t+1} 时，能依据上一时刻 t 已算得的后验概率密度函数 $p(x_t|y_{1:t})$ 直接计算新的 $p(x_{t+1} \mid y_{1:t+1})$，而不需要从头开始依据所有的累计观测数据 $y_{1:t+1}$ 重新开始计算——这样可以避免随着观测数据的累积，而造成计算量越来越大。下面我们看一下该后验概率 $p(x_t \mid y_{1:t})$ 是如何通过隐 Markov 模型传播为 $p(x_{t+1} \mid y_{1:t+1})$ 的。

依据贝叶斯公式，我们可将 $p(x_{t+1}, y_{t+1} \mid y_{1:t})$ 写为以下两种形式：

$$\begin{aligned}
p(x_{t+1}, y_{t+1} \mid y_{1:t}) &= p(x_{t+1} \mid y_{t+1}, y_{1:t})p(y_{t+1} \mid y_{1:t}) \\
&= p(y_{t+1} \mid x_{t+1}, y_{1:t})p(x_{t+1} \mid y_{1:t})
\end{aligned}$$

由于 v_t 是白噪声，所以 $t+1$ 时刻下的观测 y_{t+1} 与历史观测 $y_{1:t}$ 相互独立，$p(y_{t+1} \mid x_{t+1}, y_{1:t}) = p(y_{t+1} \mid x_{t+1})$。我们可以利用上述两式与 w_k 是白噪声的假设，将后验概率分布 $p(x_{t+1} \mid y_{1:t+1})$ 写为：

$$\begin{aligned}
p(x_{t+1} \mid y_{1:t+1}) &= \frac{p(x_{t+1}, y_{t+1} \mid y_{1:t})}{p(y_{t+1} \mid y_{1:t})} = \frac{p(y_{t+1} \mid x_{t+1})p(x_{t+1} \mid y_{1:t})}{p(y_{t+1} \mid y_{1:t})} \\
&= \frac{p(y_{t+1} \mid x_{t+1})}{p(y_{t+1} \mid y_{1:t})} \int p(x_{t+1}, x_t \mid y_{1:t})dx_t \\
&= \frac{p(y_{t+1} \mid x_{t+1})}{p(y_{t+1} \mid y_{1:t})} \int p(x_{t+1} \mid x_t, y_{1:t})p(x_t \mid y_{1:t})dx_t \\
&= \frac{p(y_{t+1} \mid x_{t+1})}{p(y_{t+1} \mid y_{1:t})} \int p(x_{t+1} \mid x_t)p(x_t \mid y_{1:t})dx_t \qquad (1)
\end{aligned}$$

值得注意的是，$p(y_{t+1} \mid y_{1:t})$ 只是一个归一化常数，可以通过以下式子计算：

$$\begin{aligned} p(y_{t+1} \mid y_{1:t}) &= \int\int p(y_{t+1} \mid x_{t+1}, x_t, y_{1:t}) p(x_{t+1} \mid x_t, y_{1:t}) p(x_t \mid y_{1:t}) dx_t dx_{t+1} \\ &= \int\int p(y_{t+1} \mid x_{t+1}) p(x_{t+1} \mid x_t) p(x_t \mid y_{1:t}) dx_t dx_{t+1} \end{aligned}$$

(1)表征了后验概率分布是如何从 $p(x_t \mid y_{1:t})$ 传递到 $p(x_{t+1} \mid y_{1:t+1})$ 的。对于线性隐 Markov 模型和高斯分布来说，(1)可以被极大程度地简化，成为 Kalman 滤波器。

3 Kalman 滤波器的数学原理

为了授课方便，这里对于 Kalman 滤波器的数学原理仅从噪声服从高斯分布的情况推导。事实上 Kalman 本人对于该滤波器的推导并不是基于高斯分布的后验概率，而是基于 Hilbert 空间下的投影原理。感兴趣的同学可以自行查看。

假设现在有下列隐 Markov 模型：

$$\begin{aligned} x_{t+1} &= A_t x_t + B u_t + w_t \\ y_t &= C_t x_t + v_t, \end{aligned}$$

其中 $x_t \in \mathbb{R}^n$ 为系统的状态，$y_t \in \mathbb{R}^p$ 为对系统的观测（或系统的输出），$w_t \sim \mathcal{N}(0, \Sigma_w)$ 为过程噪声，$v_t \sim \mathcal{N}(0, \Sigma_v)$ 为观测噪声；w_t 和 v_t 均为白噪声；初始状态的先验分布为 $x_1 \sim \mathcal{N}(m_1, P_1)$。

依据上述假设，可知

$$\begin{aligned} p(x_{t+1} \mid x_t, u_t) &= \mathcal{N}(A_t x_t + B u_t, \Sigma_w) \\ p(y_t \mid x_t) &= \mathcal{N}(C_t x_t, \Sigma_v). \end{aligned}$$

我们首先证明以下两个引理，它们将在 Kalman 滤波器的推导中扮演重要的角色。

Lemma 1 *若 $x \in \mathbb{R}^{d_x} \sim \mathcal{N}(m, P)$，$y \in \mathbb{R}^{d_y} \mid x \sim \mathcal{N}(Hx + u, R)$，则 x 和 y 的联合分布服从*

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m \\ Hm + u \end{bmatrix}, \begin{bmatrix} P & PH^T \\ HP & HPH^T + R \end{bmatrix} \right)$$

Proof:

$$\begin{aligned} & p(x, y) = p(x) p(y \mid x) \\ &= \underbrace{\frac{1}{(2\pi)^{(d_x + d_y)/2} \sqrt{\det(P) \det(R)}}}_{\gamma} e^{-\frac{1}{2}(x-m)^T P^{-1} (x-m) - \frac{1}{2}(y - Hm - u)^T R^{-1} (y - Hm - u)} \\ &= \gamma \cdot exp\left(-\frac{1}{2} \begin{bmatrix} (x-m)^T & (y - Hx - u)^T \end{bmatrix} \begin{bmatrix} P^{-1} & \\ & R^{-1} \end{bmatrix} \begin{bmatrix} x - m \\ y - Hx - u \end{bmatrix} \right) \\ &= \gamma \cdot exp\left(-\frac{1}{2} \begin{bmatrix} (x-m)^T & (-H(x-m) + y - Hm - u)^T \end{bmatrix} \begin{bmatrix} P^{-1} & \\ & R^{-1} \end{bmatrix} \begin{bmatrix} x - m \\ -H(x-m) + y - Hm - u \end{bmatrix} \right) \\ &= \gamma \cdot exp\left(-\frac{1}{2} \begin{bmatrix} (x-m)^T & (y - Hm - u)^T \end{bmatrix} \underbrace{\begin{bmatrix} I & -H^T \\ & I \end{bmatrix} \begin{bmatrix} P^{-1} & \\ & R^{-1} \end{bmatrix} \begin{bmatrix} I & \\ -H & I \end{bmatrix}}_{\Sigma^{-1}} \begin{bmatrix} x - m \\ y - Hm - u \end{bmatrix} \right) \end{aligned}$$

其中，

$$\begin{aligned}\Sigma &= \begin{bmatrix} I & \\ -H & I \end{bmatrix}^{-1} \begin{bmatrix} P^{-1} & \\ & R^{-1} \end{bmatrix}^{-1} \begin{bmatrix} I & -H^T \\ & I \end{bmatrix}^{-1} = \begin{bmatrix} I & \\ H & I \end{bmatrix} \begin{bmatrix} P & \\ & R \end{bmatrix} \begin{bmatrix} I & H^T \\ & I \end{bmatrix} \\ &= \begin{bmatrix} P & PH^T \\ HP & HPH^T + R \end{bmatrix}\end{aligned}$$

另外，

$$\det(\Sigma) = \underbrace{\det(\begin{bmatrix} I & \\ H & I \end{bmatrix})}_{=1} \det(\begin{bmatrix} P & \\ & R \end{bmatrix}) \underbrace{\det(\begin{bmatrix} I & H^T \\ & I \end{bmatrix})}_{=1} = \det(P)\det(R)$$

所以 $\gamma = \frac{1}{(2\pi)^{(d_x+d_y)/2}\sqrt{\det(\Sigma)}}$。所以命题得证。 □

Lemma 2 如果随机向量 $x \in \mathbb{R}^n$ 和 $y \in \mathbb{R}^m$ 服从以下联合高斯分布

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix} \right),$$

则

$$\begin{aligned} &x \sim \mathcal{N}(a, A), \quad y \sim \mathcal{N}(b, B) \\ &x \mid y \sim \mathcal{N}(a + CB^{-1}(y-b), A - CB^{-1}C^T), \\ &y \mid x \sim \mathcal{N}(b + C^TA^{-1}(x-a), B - C^TA^{-1}C) \end{aligned}$$

有了上述两个引理，我们现在可以在高斯白噪声的假设下推导 Kalman 滤波器的迭代式。我们不妨假设 $x_t \mid u_{1:t-1}, y_{1:t} \sim \mathcal{N}(\hat{x}_{t|t}, P_{t|t})$。由引理1可知，

$$\begin{aligned} p(x_{t+1}, x_t \mid u_{1:t}, y_{1:t}) &= p(x_{t+1} \mid x_t, u_t) p(x_t \mid u_{1:t-1}, y_{1:t}) = \mathcal{N}(A_t x_t + Bu_t, \Sigma_w)\mathcal{N}(\hat{x}_{t|t}, P_{t|t}) \\ &= \mathcal{N}\left(\begin{bmatrix} \hat{x}_{t|t} \\ A_t\hat{x}_{t|t} + Bu_t \end{bmatrix}, \begin{bmatrix} P_{t|t} & P_{t|t}A_t^T \\ A_tP_{t|t} & A_tP_{t|t}A^T + \Sigma_w \end{bmatrix} \right). \end{aligned}$$

由引理2可知，

$$p(x_{t+1} \mid u_{1:t}, y_{1:t}) = \mathcal{N}(\underbrace{A_t\hat{x}_{t|t} + Bu_t}_{\hat{x}_{t+1|t}}, \underbrace{A_tP_{t|t}A^T + \Sigma_w}_{P_{t+1|t}})$$

至此，我们就有了 Kalman 滤波器的前半部分——Prediction step：

$$\begin{aligned} \hat{x}_{t+1|t} &= A_t\hat{x}_{t|t} + Bu_t \\ P_{t+1|t} &= A_tP_{t|t}A^T + \Sigma_w. \end{aligned}$$

另一方面，依据引理1，有

$$\begin{aligned} p(x_{t+1}, y_{t+1} \mid u_{1:t}, y_{1:t}) &= p(y_{t+1} \mid x_{t+1}) p(x_{t+1} \mid u_{1:t}, y_{1:t}) \\ &= \mathcal{N}(C_tx_t, \Sigma_v)\mathcal{N}(\hat{x}_{t+1|t}, P_{t+1|t}) \\ &= \mathcal{N}\left(\begin{bmatrix} \hat{x}_{t+1|t} \\ C\hat{x}_{t+1|t} \end{bmatrix}, \begin{bmatrix} P_{t+1|t} & P_{t+1|t}C^T \\ CP_{t+1|t} & CP_{t+1|t}C^T + \Sigma_v \end{bmatrix} \right) \end{aligned}$$

进一步地，依据引理2，我们可求得后验概率分布

$$p(x_{t+1} \mid u_{1:t}, y_{1:t+1}) = \mathcal{N}\left(\hat{x}_{t+1|t+1}, P_{t+1|t+1}\right)$$

其中，

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + P_{t+1|t}C^T(CP_{t+1|t}C^T + \Sigma_v)^{-1}(y_{t+1} - C\hat{x}_{t+1|t})$$
$$P_{t+1|t+1} = P_{t+1|t} - P_{t+1|t}C^T(CP_{t+1|t}C^T + \Sigma_v)^{-1}CP_{t+1|t}.$$

至此，我们得到了 Kalman 滤波器的后半部分——Correction （Update）Step。

Kalman 滤波器通过 Prediction step 和 Correction step 交替迭代，成功地通过 $p(x_t \mid u_{1:t-1}, y_{1:t})$ 来计算 $p(x_{t+1} \mid u_{1:t}, y_{1:t+1})$。值得注意的是，该算法需要一个初始分布 $p(x_1)$，用于求 $p(x_1 \mid y_1)$，所以我们需要指定 x_1 的先验分布 $x_1 \sim \mathcal{N}(m_1, P_1)$。

4 Numpy、Scipy 使用初探

numpy 是 python 编程中常用的扩展包，可以方便的处理数组，并提供了简单的矩阵运算的功能。scipy 是基于 numpy 开发的数学工具包，支持更复杂的矩阵运算。它们将在接下来的课程学习中大量使用。

4.1 版本和安装

课程使用的镜像预装了 python2.7.12 版本，并已经安装了对应的 numpy1.11.0 和 scipy1.2.2。由于 python2 现已停止支持，最新版本的 numpy 和 scipy 在 python2 中不可用，查阅文档时请注意版本区别。

4.2 创建数组对象

numpy 的基本数据结构是 array。下面是几种常用的创建 array 方法：

```
>>> import numpy as np
>>> import scipy.linalg as lnr
>>> A_list = [
...     [1,2,3],
...     [4,5,6]
... ]
>>> A = np.array(A_list)          # 从列表创建array
>>> print A
[[1 2 3]
 [4 5 6]]
>>> # 指定array数据类型为浮点型
>>> A = np.array(A_list, dtype = np.float)
>>> print A
[[ 1.  2.  3.]
 [ 4.  5.  6.]]
>>> np.zeros((2,3))      # 创建2行3列的零矩阵
array([[ 0.,  0.,  0.],
```

```
       [ 0.,  0.,  0.]])
>>> np.ones((2,3))       # 创建全1矩阵
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> np.eye(3)            # 创建3维单位方阵
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> # 创建随机数矩阵，每个元素独立服从U[0,1]
>>> np.random.random((3,2))
array([[ 0.00783261,  0.9901787 ],
       [ 0.67345818,  0.56650474],
       [ 0.61189143,  0.53739241]])
>>> np.random.random((3,2,2)) # 创建3层2行2列高维数组
array([[[ 0.63486732,  0.01954174],
        [ 0.73169491,  0.92889407]],

       [[ 0.19185609,  0.13109882],
        [ 0.48347947,  0.15575861]],

       [[ 0.51188658,  0.53638946],
        [ 0.54709691,  0.11810446]]])
>>> ls = [1,2,3,4]
>>> np.diag(ls) # 创建对角阵
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
>>> lnr.block_diag(np.ones((2,2)), 2*np.ones((3,2)))    # 创建分块对角阵
array([[ 1.,  1.,  0.,  0.],
       [ 1.,  1.,  0.,  0.],
       [ 0.,  0.,  2.,  2.],
       [ 0.,  0.,  2.,  2.],
       [ 0.,  0.,  2.,  2.]])
>>> A.repeat((2,3), axis=0)  # 矩阵元素重复
array([[1, 2, 3],
       [1, 2, 3],
       [4, 5, 6],
       [4, 5, 6],
       [4, 5, 6]])
>>> np.tile(A,[2,3])    # 矩阵重复
array([[1, 2, 3, 1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6, 4, 5, 6],
       [1, 2, 3, 1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6, 4, 5, 6]])
```

4.3 array 常用的成员变量

array 的成员变量记录了一些基本信息。

```
>>> A = np.random.random((4,3,2))
>>> A.dtype # array数据类型
dtype('float64')
>>> A.ndim  # array维数
3
>>> A.size  # array元素个数
24
>>> A.shape # array形状
(4, 3, 2)
>>> A.data  # array存放数据的地址（类似C中的指针）
<read-write buffer for 0x7f2096e61990, size 192, offset ...>
```

4.4 常用数组操作和元素访问

```
>>> #    一元数组操作
... A = np.array([
...      [1,2,3,4],
...      [5,6,7,8],
...      [9,10,11,12]
... ])
>>> A = A.astype(np.float64)    # 更改数据类型
>>> B = A.T      # 矩阵转置
>>> B
array([[  1.,   5.,   9.],
       [  2.,   6.,  10.],
       [  3.,   7.,  11.],
       [  4.,   8.,  12.]])
>>> # 维度拆分和合并。注意:直接改变A，没有返回值
>>> A.resize(3,2,2)
>>> A
array([[[  1.,   2.],
        [  3.,   4.]],

       [[  5.,   6.],
        [  7.,   8.]],

       [[  9.,  10.],
        [ 11.,  12.]]])
>>> C = np.transpose(A, [0,2,1])    # 高维数组交换维度
>>> C
array([[[  1.,   3.],
```

```
        [  2.,   4.]],

       [[  5.,   7.],
        [  6.,   8.]],

       [[  9.,  11.],
        [ 10.,  12.]]])
>>>
>>> #    数组拼接
... D1 = np.array([[1,2,3],[4,5,6]])
>>> D2 = np.array([[7,8,9],[10,11,12]])
>>> np.stack([D1,D2],axis=0)    # 增维拼接
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
>>> np.stack([D1,D2],axis=1)
array([[[ 1,  2,  3],
        [ 7,  8,  9]],

       [[ 4,  5,  6],
        [10, 11, 12]]])
>>> np.stack([D1,D2],axis=2)
array([[[ 1,  7],
        [ 2,  8],
        [ 3,  9]],

       [[ 4, 10],
        [ 5, 11],
        [ 6, 12]]])
>>> np.concatenate([D1, D2], axis = 0)  # 同维度拼接
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
>>> np.concatenate([D1, D2], axis = 1)
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
>>>
>>> #    访问数组元素
... A = np.array([range(20)])
>>> A.resize((4,5))
>>> A
array([[ 0,  1,  2,  3,  4],
```

```
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> A[3,4]       # 访问第3行第4列的元素
19
>>> A[0,:]       # 访问第0行的所有元素
array([0, 1, 2, 3, 4])
>>> A[:,-2]      # 访问倒数第2列的元素
array([ 3,  8, 13, 18])
>>> A[1,1:4]     # 访问第1行，第1-4列的元素
array([6, 7, 8])
>>> # 把A的成员copy给B。注意:A的data成员被直接赋给B,
>>> # A和B的数据实际共用同一段地址。(浅copy)
>>> B = A
>>> D = np.copy(A)  # 将A的数据copy给D，不同址(深copy)
>>> C = B.T      # C是B的转置
>>> C[0,:] = C[1,:] # 把C第1行的值赋给第0行
>>> C            # C的值改变
array([[ 1,  6, 11, 16],
       [ 1,  6, 11, 16],
       [ 2,  7, 12, 17],
       [ 3,  8, 13, 18],
       [ 4,  9, 14, 19]])
>>> # 注意，转置不改变数据地址，只是以不同顺序访问，
>>> # 因此改变C也会改变同样使用这块地址的B
>>> B
array([[ 1,  1,  2,  3,  4],
       [ 6,  6,  7,  8,  9],
       [11, 11, 12, 13, 14],
       [16, 16, 17, 18, 19]])
>>> # 直接“=”赋值得到的B与A的data成员指向相同地址，
>>> # 因此A的值也改变
>>> A
array([[ 1,  1,  2,  3,  4],
       [ 6,  6,  7,  8,  9],
       [11, 11, 12, 13, 14],
       [16, 16, 17, 18, 19]])
>>> D   # 深copy得到的D没有改变
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

4.5 矩阵基本运算

```
# 矩阵基本运算
A = np.array(range(1,13))
A.resize((4,3))
np.mean(A,axis=0)    # 计算每行的平均值
np.min(A,axis=1)     # 计算每列的最小值
np.max(A)            # 计算整个数组的最大值
B = np.array(range(1,13))
B.resize((4,3))
A+B      # 矩阵加法。减法同
A*B      # 按位相乘，同matlab中的.*。除法同
A.dot(B.T)  # 矩阵乘法。

# 一般来说位运算需要两矩阵维度完全相同，
# 但服从广播(broadcast)规则时，运算仍合法，如下例。
A = np.array(range(24))
A.resize((4,3,2))
C = np.array(range(8))
C.resize((4,1,2))
A*C # A,C维数相同，长度不同的维度，C的长为1，运算合法
D = np.array(range(6))
D.resize((3,2))
A*D # A，D维数不同，D的维度与A的最后几个维度相同，合法
C*D # 同时应用上面两个规则，运算合法。
```

4.6 进阶矩阵运算

利用 scipy 的 linalg 模块，可以实现一些略复杂的矩阵运算。如下例。另，scipy 还有 optimize、signal 等模块，具体参阅具体参阅http://scipy.github.io/devdocs/reference/index.html。

```
import scipy.linalg as lnr
A=np.random.randint(0,10,(4,4))
r = np.linalg.matrix_rank(A)    #矩阵秩
d = lnr.det(A)           # 行列式
n = lnr.norm(A)          # 矩阵范数
n = lnr.norm(A[:,1])     # 向量范数
B = lnr.pinv(A)          # 伪逆
val, vec_r = lnr.eig(A) # 特征值和特征向量
U, s, Vh = lnr.svd(A)    # 奇异值分解
C = lnr.expm(A)          # 矩阵指数运算
```

4.7 array 数据的可视化

matplotlib 是最常用的 python 数据可视化工具。它的优点是自由度高，功能强大，但缺点是接口非常底层，代码量稍大。这里举例介绍 matplotlib 的简单应用。

```
from matplotlib import pyplot as plt
A = np.random.multivariate_normal([0,1], [[1,0.3],[0.3,1]], (1024))
fig, ax = plt.subplots(2,2)              # 绘制含2x2子图的图窗
ax[0,0].scatter(A[:,0], A[:,1])          # 散点图
ax[0,0].set_title("Scatter chart")       # 设置图标题
ax[0,1].hist2d(A[:,0], A[:,1], bins=20) # 用hist2d实现的热度图
ax[0,1].set_title("Heatmap")             #       hist可以作直方图
ax[1,0].plot(range(20), A[:20,0])        # 折线图
ax[1,0].set_title("Line chart")
ax[1,1].boxplot([A[:,0], A[:,1]])        # 箱线图
ax[1,1].set_title("Boxplot")
fig.show()                          # 展示图窗
```

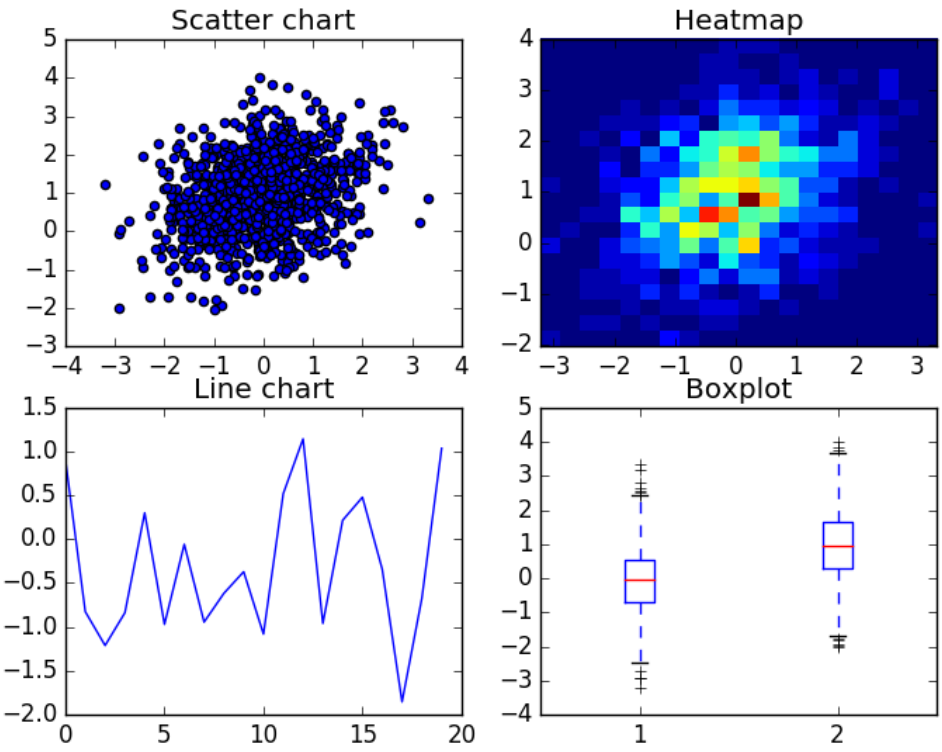

Figure 2: matplotlib 画图

5 ROS 例程：Publisher / Subscriber

本节将通过例程介绍，如何从搭建 catkin 工作空间开始，编写你自己的 ROS package，自定义一个 message 实例，并实现一对发布者（Publisher）/订阅者（Subscriber），最后通过 roslaunch 启

动他们。简便起见，本课程主要使用 python2 作为编程语言，这里不涉及 C++ 的部分。

5.1 Catkin 工作空间

Catkin 是 ROS 官方推荐的编译系统，结合了 CMake 和 Python，对 CMake 的工作流进行封装，使用更简便。相比旧的 rosbuild，catkin 提供了更好的移植性。更具体的描述可参阅http://wiki.ros.org/catkin/conceptual_overview。

Carkin 工作空间（Carkin Workspace），简单的说，是你编辑和编译代码的文件夹。只要搭好规范的文件目录结构，并在文件（如 CMakeLists.txt）中给出要求的编译规则，catkin 可以帮你“一键”编译好目录下的所有项目！关于 catkin workspace 更详细的说明可查阅http://wiki.ros.org/catkin/workspaces。但现在可以先跳过这些细节，开始搭建你的 catkin 工作空间。在终端运行以下命令

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws
$ catkin_make
```

catkin_make 是 catkin workspace 下的编译指令。初次运行时，目录下会生成 build、devel 两个文件夹，并在 src 文件夹生成 CMakeLists.txt 文件。

要使用这个新工作空间下的功能包（package），必须给终端添加一些环境变量。catkin 刚刚已经帮你准备好了脚本，继续输入命令 (以 bash 为例)：

```
$ source ./devel/setup.bash
```

检查一下环境变量是否正确：

```
$ echo $ROS_PACKAGE_PATH
```

预期大概会得到：

```
/home/ubuntu/catkin_ws/src:/opt/ros/kinetic/share
```

这是当前终端所知的所有 ros 功能包的路径。路径之间由冒号分隔。

5.2 创建新功能包（package）

接下来，在工作空间里创建一个新功能包：

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg tutorials std_msgs rospy
# catkin_create_pkg <package_name> [depend1] [depend2]
```

这里用到了新命令：catkin_create_pkg，它创建了一个名叫 tutorials 的新功能包，这个包依赖于已有的 ros 功能包：std_msgs, rospy。运行这个命令之后，你会看到新文件夹 catkin_ws/src/tutorials。其中 package.xml 文件中是该功能包的基本数据，包括名称描述、作者、证书、依赖项等。其中最重要的是 51 行开始的这一部分：

```
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
```

```
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
```

接下来编译这个新功能包

```
$ cd ~/catkin_ws
$ catkin_make
```

这样就生成了一个新功能包。例如，可以直接使用 roscd 命令访问，不必输入绝对路径:

```
$ roscd tutorials
# it is equivalent to
# cd ~/catkin_ws/src/tutorials
```

5.3 创建一对发布者 (Publisher)/订阅者 (Subscriber) 节点 (Node)

简单的说，节点 (Node) 就是 ROS 功能包中的一个可执行文件。Topic 是节点之间信息交流的方式之一。例如，对一个 Topic，节点 A，B 注册成为该 Topic 的发布者 (Publisher)，他们都可以不定时的向 Topic 发送格式化的消息 (message)；节点 C，D 可以订阅 (Subscribe) 该 Topic，每当 Topic 上出现新 message，C，D 都会收到通知，并调用各自的回调函数 (Callback Function) 处理新 message。

接下来我们将在刚刚创建的功能包中编写一对发布者/订阅者节点。

5.3.1 发布者节点 (Publisher Node)

```
$ roscd tutorials
$ mkdir scripts
$ cd scripts
$ touch str_talker.py
```

接下来编辑 str_talker.py，代码如下。也可以在例程中找到。

```
1  #!/usr/bin/env python
2  # license removed for brevity
3  import rospy
4  from std_msgs.msg import String
5
6  def talker():
7      pub = rospy.Publisher('chatter',\
8          String, queue_size=10)
9      rospy.init_node('talker', anonymous=True)
10     rate = rospy.Rate(10) # 10hz
11     while not rospy.is_shutdown():
12         hello_str = "hello world %s"%rospy.get_time()
13         rospy.loginfo(hello_str)
14         pub.publish(hello_str)
15         rate.sleep()
16
```

```
17 if __name__ == '__main__':
18     try:
19         talker()
20     except rospy.ROSInterruptException:
21         pass
```

逐行解释一下代码的含义，

```
3 import rospy
4 from std_msgs.msg import String
```

调用 ROS 提供的 python 库，所有用 python 实现的 ROS Node 都会调用该库。std_msgs.msg 提供了一些标准的 message 格式，这里使用 std_msg/String 格式的 message。

```
7     pub = rospy.Publisher('chatter',\
8         String, queue_size=10)
```

接下来进入主函数 talker()，这段代码表示这个节点向名叫 “chatter” 的 Topic 发送 message，message 格式是 “String”，queue_size 表示队列长度，当订阅者反映太慢时，队列不会堆积超过 10.

```
9     rospy.init_node('talker', anonymous=True)
```

上面这条命令初始化了这个节点，他的名字定为 talker，anonymous=True 通过给节点加随机后缀名的方式保证节点名的唯一性。

```
10    rate = rospy.Rate(10) # 10hz
11    while not rospy.is_shutdown():
12        hello_str = "hello world %s" % rospy.get_time()
13        rospy.loginfo(hello_str)
14        pub.publish(hello_str)
15        rate.sleep()
```

第 10 行和第 15 行的命令合起来保证该循环以 10hz 的频率运行（按照 ROS 的虚拟时钟）。while 循环检查程序是否退出，若未退出则执行循环。rospy.loginfo(hello_str) 将信息打印在终端中，写入该节点的 log file，同时发给 rosout 节点。rosout 节点可以配合 rqt_console 工具，方便开发者 debug。pub.publish(hello_str) 则是向 “chatter” Topic 发布 message。

```
17 if __name__ == '__main__':
18     try:
19         talker()
20     except rospy.ROSInterruptException:
21         pass
```

当运行节点时，调用主函数 talker()。异常 rospy.ROSInterruptException 可能由 rospy.sleep() 或 rospy.Rate.sleep() 抛出，表示节点关闭（例如你按下 Ctrl+C）。

5.3.2 订阅者节点 (Subscriber)

接下来创建订阅者节点。

```
$ roscd tutorials/scripts
$ touch str_listener.py
```

编辑 str_listener.py 如下，例程中也可以找到：

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() \
        + "I heard %s", data.data)

def listener():
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("chatter", String, callback)
    rospy.spin()

if __name__ == '__main__':
    listener()
```

订阅者的代码中只有两行是新功能。

```
    rospy.Subscriber("chatter", String, callback)
```

该语句表示该节点订阅 “chatter” Topic，message 类型是 String，当收到 message 时，调用 callback 函数，并以收到的 message 作为 callback 的第一个参数。

```
    rospy.spin()
```

该语句保证订阅者节点在后台持续运行并接受处理 message。

5.3.3 编译节点并测试

写好发布者/订阅者的程序，先修改文件权限，再编译工作空间。

```
$ chmod +x ./str_listener.py    # 给文件以执行权限
$ chmod +x ./str_talker.py
$ cd ~/catkin_ws
$ catkin_make
```

打开 3 个终端窗口，分别在里面运行：

```
$ ## ===== Terminal No.1 =======
$ cd ~/catkin_ws
$ source ./devel/setup.bash
$ roscore
```

```
$ ## ===== Terminal No.2 =======
$ cd ~/catkin_ws
```

```
$ source ./devel/setup.bash
$ rosrun tutorials str_talker.py
```

```
$ ## ===== Terminal No.3 =======
$ cd ~/catkin_ws
$ source ./devel/setup.bash
$ rosrun tutorials str_listener.py
```

你会在终端 No.2 No.3 中分别看到发布者和订阅者输出的 helloworld。

5.3.4 定义自己的 message 格式

有时，ROS 提供的 message 格式并不能满足我们的需求，我们需要定义自己的 message 格式。
首先创建一个.msg 文件

```
$ roscd tutorials
$ mkdir msg
$ touch ./msg/person.msg
```

编辑该文件如下：

```
string name
uint8 age
```

文件每行是一个成员，前半段是成员类型，后半段是成员名。msg 支持的成员类型有：

- int8, int16, int32, int64 (还有 uint*)
- float32, float64
- string
- time, duration
- other msg files(其他文件定义的 msg 类型)
- array[](变长数组), array[C](定长数组)

修改功能包的 package.xml 文件（tutorials/package.xml），添加依赖：

```
  <build_depend>message_generation</build_depend>
  <exec_depend>message_runtime</exec_depend>
```

修改 package 的 CMakeLists.txt(tutorials/CMakeLists.txt)，修改 find_package 部分，改成这样：

```
find_package(catkin REQUIRED COMPONENTS
  rospy
  std_msgs
  message_generation
)
```

修改 catkin_package 部分，改成这样：

```
catkin_package(
  ...
  CATKIN_DEPENDS message_runtime ...
  ...)
```

找到 add_message_files 部分，取消注释，改成这样：

```
add_message_files(
  FILES
  person.msg
)
```

最后找到 generate_messages 部分，取消注释，变成这样：

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

这样就完成了。你可以使用 rosmsg 命令查询一下，确认你的 message 类型成功创建了。

```
$ rosmsg show person
```

5.4 使用 launch 文件快速启动多个节点

回忆刚刚测试发布者/订阅者节点时，我们启动了三个终端分别运行 roscore 和两个节点，非常繁琐。ROS 项目中常常有许多节点需要启动，这时我们可以编写 launch 文件，一次性启动所有节点（也会自动启动 roscore）。

顺便，我们可以测试一下刚刚定义的 msg 类型。创建文件：

```
$ touch ./scripts/per_talker.py
$ touch ./scripts/per_listener.py
```

编辑 per_talker.py 如下，例程中也可以找到：

```
1  #!/usr/bin/env python
2
3  import rospy
4  from tutorials.msg import person
5  def talker():
6      pub = rospy.Publisher('people', \
7          person, queue_size = 10)
8      rospy.init_node('talker', anonymous=True)
9      rate = rospy.Rate(1)
10     while not rospy.is_shutdown():
11         msg = person()
12         msg.name = "Ice"
13         msg.age = 18
14         rospy.loginfo("send Name=Ice, age=18.")
```

```
        pub.publish(msg)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

编辑 per_listener.py 如下，例程中也可以找到：

```
#!/usr/bin/env python

import rospy
from tutorials.msg import person

def callback(data):
    rospy.loginfo(rospy.get_caller_id() +\
        "heard Name=%s"%data.name)

def listener():
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber('name', person, callback)
    rospy.spin()

if __name__ == '__main__':
    listener()
```

注意到，per_talker 发布 Topic 名为 people，而 per_listener 订阅 Topic 名为 name。这个问题我们可以在 launch 文件中解决。

首先创建 launch 文件：

```
$ roscd tutorials
$ mkdir launch
$ touch ./launch/nodes.launch
```

编辑 nodes.launch 如下，例程中也可以找到

```
<launch>

  <group ns="namespace1">
    <node pkg="tutorials" name="talker" type="str_talker.py"/>
    <node pkg="tutorials" name="listener" type="str_listener.py"/>
  </group>

  <group ns="namespace2">
    <node pkg="tutorials" name="talker" type="per_talker.py">
    <remap from="people" to="name"/>
    </node>
    <node pkg="tutorials" name="listener" type="per_listener.py"/>
```

```
  </group>

</launch>
```

最后重新编译所有节点，并测试：

```
$ roscd tutorials
$ chmod +x ./scripts/per_listener.py # 给文件以执行权限
$ chmod +x ./scripts/per_talker.py
$ cd ~/catkin_ws
$ catkin_make
$ roslaunch tutorials nodes.launch # 使用roslaunch启动
```

6 习题

1. 证明 Lemma 2。
2. 运用 ROS 的消息发布和接收机制，编写程序，实现小乌龟的如下运动：
   (a) 沿矩形路径运动，
   (b) 沿圆周路径运动。