KALMAN FILTER

* Linear 한 gaussian distribution을 따르는 현상의 노이즈 필터링에 적합
* 하지만 CSI 가 가우시안 분포를 꼭 따른다고 생각 할 수 없다
* 이러한 non-linear 경우는 EKF 즉 Extended Kalman filter를 쓴다
* 기존의 칼만 필터와의 차이점은 기본적인 칼만 필터는

01. Kalman 필터는 어떤 때에 쓸까요?

     - 잡음이 포함되어 있는 선형(linear, 변수가 일차이고 곱과 합으로 이루어져 있는)으로 돌아가는 원리를 가진 곳에서 그 상태(state)를 추적하는 재귀(같은 것의 파라미터를 업데이트하고 반복해서 같은 것이 수행되는)하는 곳에 쓰여요.

02. 구체적으로 어떤 환경에서 쓰이는 거죠?

     - 시간에 따라 측정한 데이터를 기반으로 쓰여요. '01번'에서 재귀적으로 쓰인다고 했는데, 10시 정각에 측정한 값도 쓰고, 10시 1분에 측정한 값도 쓴다는 것이예요. 이렇게 하면 딱 10시에 측정한 값만 이용하는 것 보다 우리가 예측하려고 하는 상태(state)값을 더 정확하게 알 수 있다는 장점이 있어서 이렇게 쓰는 거예요.

03. Kalman 필터의 수식이 엄청 복잡하고 어렵던데 왜 이런가요?

     - 칼만필터는 우리 인간의 사고와 비슷해서 그래요. **관찰하고, 예상해보고, 관찰값과 예상값을 연관시켜서 예상했던 값을 채점해보고**, 다시 관찰하고, 예상하는 것이 수반되거든요. 그걸 수식으로 표현하려다 보니 복잡해 보이지만, 예측/관찰/업데이트로 이루어진 틀이 정해져 있는 수식들이라고 칼만필터를 이해해 보세요.

04. 그럼 지금까지 얘기한 것을 그림으로 간단히 그려 놓으신건 없으세요?

     - 제가 그리진 않았지만, wiki에 좋은 그림이 있어요. '39번' 포스팅과 밑의 그림을 잘 째려보세요. 지금껏 얘기한 것들이 맞아들어가니까요.

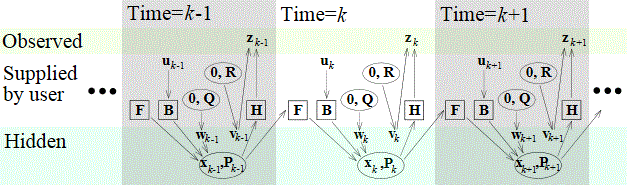


그림 1 : 칼만필터의 이해를 돕는 그림

Hidden 이라고 되어 있는 layer(3행 전체)의 x가 우리가 알고자 하는 state인 것이고, observed라는 layer(1행 전체)가 측정(관찰)한 값이 z가 되는 것이고, 우리 공학인들이 해당 모델들을 설계한 것들이 Supplied by user라는 부분이예요.

인용 : <https://en.wikipedia.org/wiki/Kalman_filter>

05. 그럼 kalman filter를 사용할 때 주의점은 있나요?

     - 초기 상태와 각 잡음 변수는 모두 상호 독립이라는 가정을 하고 있어요. 사실 대다수의 경우, 실제 시스템(우리가 예측하려는 여러 모델들)이 선형적이지 않아서, 모두 상호 독립이라는 가정에 맞질 않아요. 이런 특성들이 강할 수록 칼만 필터로 예측하는 결과의 정확도가 많이 낮아지게 되고 심각한 경우 엉뚱한 결과를 가져 올 수 도 있어요. 그래서 비선형적인 부분이 강한 모델은 EKF(Extended Kalman Filter)라는 비선형에도 대략적으로 활용할 수 있는 사용해요.

Reference: <https://www.matec-conferences.org/articles/matecconf/pdf/2016/19/matecconf_iccae2016_03002.pdf>

<https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/07-Kalman-Filter-Math.ipynb>

https://github.com/milsto/robust-kalman

Q-matrix ( using white noise assumption here)

Skeleton codes

Extended Kalman filter(non - linear)

|  |
| --- |
| """ |
|  | Robust Kalman filter |
|  |  |
|  |
|  |
|  |  |
|  | import numpy as np |
|  | from scipy.optimize import minimize |
|  | from .utils import HuberScore |
|  |  |
|  |  |
|  | class RobustKalman(): |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  | def \_\_init\_\_(self, F, B, H, x0, P0, Q0, R0, use\_robust\_estimation=False, use\_adaptive\_statistics=False, robust\_score=HuberScore(delta=1.5)): |
|  | """Initialize robust Kalman. All input matrices are coppied. |
|  |  |
|  | Args: |
|  | F: State transition matrix |
|  | B: Input transition matrix (may be None if model has no inputs) |
|  | H: Observation matrix |
|  | x0: Initial state vector |
|  | P0: Initial state covariance matrix |
|  | Q0: (Initial) state noise covariance |
|  | R0: (Initial) observation noise covariance |
|  | use\_robust\_estimation: True if robust estimation procedure should be used |
|  | use\_adaptive\_statistics: True if adaptive robust estimation of noise variance should be used |
|  | robust\_score: Score function for robust estimation. (1.5)-Huber is the default. |
|  | """ |
|  | self.F = F.copy() |
|  | self.B = B.copy() if B is not None else None |
|  | self.H = H.copy() |
|  | self.x = x0.copy() |
|  | self.P = P0.copy() |
|  | self.Q = Q0.copy() |
|  | self.R = R0.copy() |
|  |  |
|  | self.use\_robust\_estimation = use\_robust\_estimation |
|  | self.use\_adaptive\_statistics = use\_adaptive\_statistics |
|  |  |
|  | # Used for adaptive noise estimation |
|  | self.history\_inovation = list() |
|  | self.r\_mean\_est = 0.0 |
|  | self.r\_var\_est = 0.0 |
|  |  |
|  | self.robust\_score = robust\_score |
|  |  |
|  | def time\_update(self, inputs=None): |
|  | """ |
|  | Time propagation of the system model. |
|  |  |
|  | Args: |
|  | inputs: Model inputs if any. |
|  |  |
|  | """ |
|  | if inputs is None: |
|  | self.x = np.matmul(self.F, self.x) |
|  | else: |
|  | self.x = np.matmul(self.F, self.x) + np.matmul(self.B, inputs) |
|  |  |
|  | self.P = np.matmul(np.matmul(self.F, self.P), self.F.T) + self.Q |
|  |  |
|  | def measurement\_update(self, measurements): |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  | # Residual or inovation |
|  | self.inovation = measurements - np.matmul(self.H, self.x) |
|  |  |
|  | # Inovation covariance matrix |
|  | Pinov = np.matmul(np.matmul(self.H, self.P), self.H.T) + self.R |
|  |  |
|  | # Kalman gain K = Pxy \* Pinov^-1 |
|  | K = np.matmul(np.matmul(self.P, self.H.T), np.linalg.inv(Pinov)) |
|  |  |
|  | if self.use\_robust\_estimation: |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  | epsilon\_covariance = np.bmat([[self.P, np.zeros((self.P.shape[0], self.R.shape[1]))], |
|  | [np.zeros((self.R.shape[0], self.P.shape[1])), self.R]]) |
|  |  |
|  |  |
|  | S = np.linalg.cholesky(epsilon\_covariance) |
|  | Sinv = np.linalg.inv(S) |
|  |  |
|  |
|  | Y = np.matmul(Sinv, np.vstack((self.x, measurements))) |
|  |  |
|  |  |
|  |  |
|  | X = np.matmul(Sinv, np.vstack((np.eye(self.x.shape[0]), self.H))) |
|  |  |
|  |
|  |
|  |
|  |
|  | res = minimize(lambda xx: self.\_m\_estimate\_criterion(xx, Y, X), self.x, method='nelder-mead') |
|  |  |
|  | self.x = res.x[np.newaxis].T |
|  | else: |
|  | # Linear state update |
|  | self.x = self.x + np.matmul(K, self.inovation) |
|  |  |
|  |
|  |
|  |
|  | self.P = self.P - np.matmul(np.matmul(K, self.H), self.P) |
|  |  |
|  | if self.use\_adaptive\_statistics: |
|  | assert self.R.shape == (1, 1), 'Current implementation for robust variance estimation tested only for ' \ |
|  | 'models with one observable.' |
|  | self.history\_inovation.append(self.inovation) |
|  | if len(self.history\_inovation) < 6: |
|  | self.r\_mean\_est = 0.0 |
|  | self.r\_var\_est = self.R[0, 0] |
|  | else: |
|  | # Adaptive estimate of R |
|  | r\_arr = np.array(self.history\_inovation, dtype=np.float32) |
|  | d = np.median(np.fabs(r\_arr - np.median(r\_arr)) / 0.6745) |
|  |  |
|  | self.r\_mean\_est = minimize(lambda xx: self.\_m\_estimate\_r\_criterion(xx, r\_arr, d), self.history\_inovation[-1], method='nelder-mead').x |
|  | self.r\_var\_est = d\*\*2 - np.matmul(np.matmul(self.H, self.P), self.H.T) |
|  |  |
|  | self.R[0, 0] = self.r\_var\_est |
|  |  |
|  | @property |
|  | def current\_estimate(self): |
|  | return self.x |
|  |  |
|  | @property |
|  | def current\_estimate\_covariance(self): |
|  | return self.P |
|  |  |
|  | @property |
|  | def current\_inovation(self): |
|  | return self.inovation |
|  |  |
|  | def \_m\_estimate\_criterion(self, x, Y, X): |
|  | """Criterion for robust state estimation""" |
|  | crit = 0.0 |
|  | for i in range(Y.shape[0]): |
|  | crit += self.robust\_score.evaluate(Y[i, :] - np.matmul(X[i, :], x)) |
|  | #crit += (Y[i, :] - np.matmul(X[i, :], x))\*\*2 |
|  |  |
|  | return crit |
|  |  |
|  | def \_m\_estimate\_r\_criterion(self, x, r\_est\_arr, d): |
|  |  |
|  | crit = 0.0 |
|  | for i in range(len(r\_est\_arr)): |
|  | crit += self.robust\_score.evaluate((r\_est\_arr[i] - x) / d) |
|  |  |
|  | return crit |

PCA (주성분 분석)

**PCA란 한마다로 말하면 입력 데이터들의 공분산 행렬(covariance matrix)에 대한 고유값분해(eigendecomposition)로 볼 수 있다.**  
이 때 나오는 고유벡터가 주성분 벡터로서 데이터의 분포에서 분산이 큰 방향을 나타내고,   
대응되는 고유값(eigenvalue)이 그 분산의 크기를 나타낸다.  
  
-결국 데이터 집합의 공분산 행렬을 구한 후, 이의 고유값 λ 과, 고유 벡터  u 를 계산하면 이것이 최대 분산을 가진 값이 된다.  
-차원 축소를 통해 최소 차원의 정보로 원래 차원의 정보를 모사(approximate)하려는 작업.  
원래 있던 데이터로 선형조합을 한다.  
-차원 축소는 고유값이 높은 순으로 정렬해서, 높은 고유값을 가진 고유벡터만으로 데이터 복원 (주성분)  
  
-낮은 고유값을 가진다는 의미는 해당 고유벡터로는 변화(scale)가 적다라는 의미이기 때문

**PCA의 목표  
변환 결과인 차원축소 벡터가 원래의 벡터와 가장 유사하게 되는 W 값을 찾는 것이다.**

**사용도 :**  
-PCA가 영상인식에 활용되는 대표적인 예는 얼굴인식(face recognition)이다.   
-차원축소를 하여 어떠한 지표를 갖고싶을때  
-경제 index 지수. 코스피지수 등  
- 정규화,  
- 노이즈(noise)에 대해 좀더 생각해 보면, 앞서 말했듯이 PCA는 개별 데이터에 대한 분석이 아니라 전체 데이터에 대한 집합적 분석 도구  
- 다중공선성이 존재할 경우, 상관도가 높은 변수를 축소  
- 연관성 높은 변수를 축소하여 연산속도 및 결과치 개선  
- 기계에서 나오는 다양한 센서데이터를 주성분분석으로 차원축소 후 시계열로 분포나 추세를 분석하여 고장 징후를 사전에 파악한다.

Skeleton code

from sklearn.decomposition import PCA

X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])

pca = PCA(n\_components=2)

pca.fit(X)

Z = pca.transform(X)

w, V = np.linalg.eig(pca.get\_covariance())

V.T.dot(X.T).T

Process figure

1. CSI extraction -> Kalman filter -> PCA -> LSTM (RNN base)
2. CSI\_NET (CNN base)