Statespace

Noah Smith

Dec 21, 2022

CONTENTS:

this project focuses on reference problems in kalman, sigma-point, and particle processors - in particular, numpy and eigen implementations, using classic vector and matrix representations so things look like what's in the literature.

CHAPTER

ONE

INSTALL

have switched to a container build environment - reproducible using the dockerfile. everything needed is there and can be reproduced for a local environment. docker image is available on docker hub.

CHAPTER

TWO

PROCESSORS

2.1 kalman.py

baseline extended kalman filters. both the standard textbook form, and the ud factorized form.

class statespace.processors.kalman.**Kalman**(**args*, ***kwargs*)

classical extended kalman filter

ekf(*model*)

basic form

ekfud(*model*)

UD factorized form

observational(*xin*, *Uin*, *Din*, *H*, *obs*, *R*, *yhat*)

bierman observation update

temporal(*xin*, *Uin*, *Din*, *Phi*, *Gin*, *Q*)

thornton temporal update

udfactorize(*M*)

UD factorization

2.2 sigmapoint.py

sigma-point sampling kalman filters. this is the ukf or unscented kalman filter, or 'modern' kalman filtering. it's essentially somewhere between a classical kalman filter, where uncertainty is represented as gaussian, and a particle filter, where uncertainty has arbitrary shape. here uncertainty is deterministically sampled at a small number of points, the sigma points or sigma particles. in a particle filter the number and role of the particles are increased.

class statespace.processors.sigmapoint.**SigmaPoint**(**args*, ***kwargs*)

modern sigma-point deterministic sampling kalman filter

choldowndate(*R*, *z*)

cholesky downdate

cholupdate(*R*, *z*)

cholesky update

observational(*x*, *o*, *X*, *Y*, *W*, *Xtil*, *Ytil*, *Pxy*, *S*, *Sobs*)

cholesky factorized observational update

spf(*model*)

sigma-point deterministic sampling kalman filter

spfcholesky(*model*)

cholesky factorized sigma-point sampling kalman filter

temporal(*x*, *XY*, *W*, *Xtil*, *S*, *Sproc*, *u*)

cholesky factorized temporal update

2.3 particle.py

particle filters, sequential monte carlo sampling processors. sampling here is random, not deterministic as in the sigma-point processor. and the idea of resampling and growing new particles comes to the fore. the particles are random and new ones can be introduced freely.

class statespace.processors.particle.**Particle**(**args*, ***kwargs*)

particle filter, sequential monte carlo processor

pf(*model*)

basic sampling-importance-resampling (SIR) particle filter, bootstrap particle filter, condensation particle filter, survival of the fittest algorithm

CHAPTER

THREE

MODELS

will probably be moving towards a higher-level statespace model, encompassing specific lower-level models - possibly something involving a translator / converter / adaptor… the models here are already an extrememly primitive form of that - making them as similar as possible from the perspective of the classical, modern, particle processors. we can think about these becoming specific cases of something more fundamental.

3.1 basemodel.py

placeholder for what could grow to become a higher-level statespace model - with individual models inheriting and overriding.

class statespace.models.basemodel.**BaseModel**

base model

3.2 onestate.py

a simple as possible one-state example with non linear temporal and observation updates. it's a common example in the candy and jazwinisky books. based on real world reentry vehicle tracking.

class statespace.models.onestate.**Onestate**

one-state reference model

3.3 threestate.py

three-state extension of the the one-state model. non linear temporal and observation updates.

class statespace.models.threestate.**Threestate**

three-state reference model

3.4 bearingsonly.py

the bearings only problem has some interesting history. it's basically about being on a sub. your sub is travelling along steadily and you begin hearing the sound of a ship at some bearing. over time and as the bearing changes, you can estimate the relative position and velocity of the ship. at some point you make a course change for your sub to pursue the ship.

class statespace.models.bearingsonly.**BearingsOnly**

bearings-only tracking problem

PYTHON MODULE INDEX

S

INDEX