ZÁPADOČESKÁ UNIVERZITA V PLZNI

A software framework and tool for nonlinear state estimation

Ondřej Straka, Miroslav Flídr, Jindřich Duník and Miroslav Šimandl

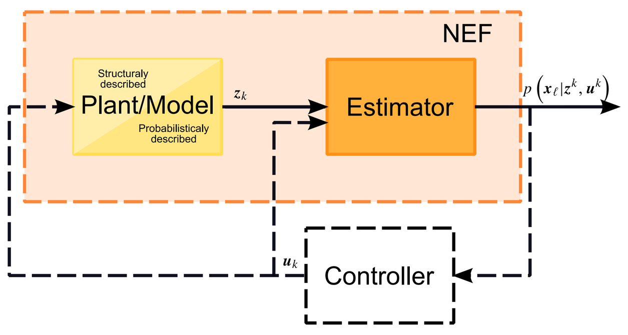
Department of Cybernetics & Research Centre Data - Algorithms - Decision making
Faculty of Applied Sciences, University of West Bohemia, Czech Republic

DAR

Main goal of the Nonlinear Estimation Framework (NEF)

Provide a software framework designed for nonlinear state estimation of discrete time stochastic dynamic systems.

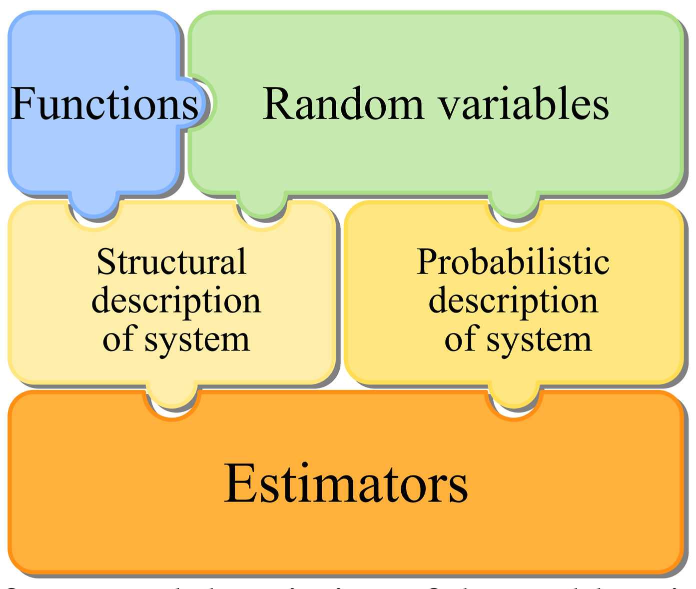
Tasks supported by NEF
- description and simulation of a plant/model,
- estimation of the state of the model,
- providing estimates in terms of probability density functions.



Nonlinear Estimation Framework features

- highly modular and extensible

Figure: NEF components and their relation



- designed with support for natural description of the problem in mind
  - enables both the **structural** and **probabilistic** description of a system
  - supports specification of **time-varying** systems
- fast and easy estimation experiment setup
- facilitates implementation of **filtering**, **prediction** and **smoothing** tasks
- implements many of the popular nonlinear state estimators

Local methods	
Estimator	Num. stable version
(Extended) Kalman filter	✓ (UD)
Unscented Kalman filter	✓
Divided difference filter 1st order	✓
Divided difference filter 2nd order	-
Iteration filter (with any other local filter)	-

Global methods	
Estimator	Note
Particle filter	sampling densities: prior, optimal, EKF, auxiliary (point and functional)
Gaussian sum filter	can utilize most of the local filters

- NEF is available upon request for free for noncommercial use

Description of the NEF components

A function of state, input, noise and time is a fundamental element

Functions in structural description

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k, k)$$
$$\boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k, k)$$

Functions in probabilistic description

$$p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k) = \mathcal{N}\{\boldsymbol{x}_{k+1} : \boldsymbol{\mu}_x(\boldsymbol{x}_k, \boldsymbol{u}_k, k), \boldsymbol{Q}(\boldsymbol{x}_k, \boldsymbol{u}_k, k)\}$$
$$p(\boldsymbol{z}_k|\boldsymbol{x}_k) = \mathcal{N}\{\boldsymbol{z}_k : \boldsymbol{\mu}_z(\boldsymbol{x}_k, \boldsymbol{u}_k, k), \boldsymbol{R}(\boldsymbol{x}_k, \boldsymbol{u}_k, k)\}$$

Provided classes for description of multivariate functions	
`nefHandleFunction`	general function described by MATLAB handle function
`nefLinFunction`	linear function
`nefConstFunction`	constant function

`nefHandleFunction` : the most useful and common way of describing the functions

❶ create regular or handle function in MATLAB
```
mFun = @(x,u,v,t) atan(x(3)/x(1));
```
❷ create `nefHandleFunction` instance with appropriate parameters
```
fun = nefHandleFunction(mFun,[4 0 0 0]);
```

Both descriptions require ways to describe random variables

What does NEF offer for random variables description?
- classes describing many multivariate and univariate probability density functions
- natural way of the creation of random variable

Description of uniform random variable

$$p(a) = \mathcal{U}(-2, 4)$$
```
a=nefUniformRV(-2,4);
```

Description of Gaussian transition pdf

$$p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k) = \mathcal{N}\{\boldsymbol{x}_{k+1} : F\boldsymbol{x}_k + G\boldsymbol{u}_k, Q\}$$
```
xMean=nefLinFunction(F,G,[]);
px=nefGaussianRV(xMean,Q);
```

Plant/model can be described in two ways

Structural description
```
nefEqSystem(f,h,pw,pv,px0)
```
- `f` … state transition function
- `h` … measurement function
- `pw` … state noise pdf
- `pv` … measurement noise pdf
- `px0` … prior state pdf

Probabilistic description
```
nefPDFSystem(px,pz,px0)
```
- `px` … transition pdf
- `pz` … measurement pdf
- `px0` … prior state pdf

The estimation experiment can be completed in two simple steps

Estimation experiment execution

❶ create an instance of estimator class with appropriate constructor parameters
❷ execute the estimation task
a) automatically using `estimate` method
b) calling methods `timeUpdate`, `measurementUpdate` and `smoothUpdate` in the right order

Simple estimation experiment example

❶ create instance of UKF filter
```
filter = nefUKF(model);
```
❷ process data and provide $p(\boldsymbol{x}_k|\boldsymbol{z}^k)$
```
pdfs = estimate(filter,z,[]);
```

Estimation experiment design for a structurally described system

Let the state of the following system be estimated

$$\boldsymbol{x}_{k+1} = \begin{pmatrix} x_{1,k+1} \\ x_{2,k+1} \end{pmatrix} = \begin{pmatrix} x_{2,k} \cdot x_{1,k} \\ x_{2,k} \end{pmatrix} + \boldsymbol{w}_k,$$
$$z_k = (1, 0)\, \boldsymbol{x}_k + v_k,$$

where $\boldsymbol{x}_k$ denotes the state to be estimated and z_k denotes the measurement. The stochastic quantities $\boldsymbol{w}_k$ and v_k are described by the following pdf's

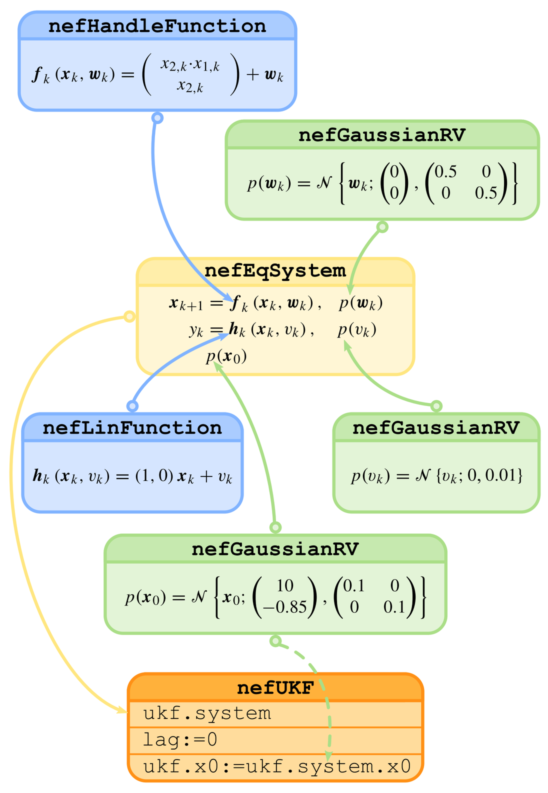
$$p(\boldsymbol{w}_k) = \mathcal{N}\left\{\boldsymbol{w}_k; \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\right\},$$
$$p(v_k) = \mathcal{N}\{v_k; 0, 0.01\},$$

and the pdf of the initial state $\boldsymbol{x}_0$ is

$$p(\boldsymbol{x}_0) = \mathcal{N}\left\{\boldsymbol{x}_0; \begin{pmatrix} 10 \\ -0.85 \end{pmatrix}, \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}\right\}.$$

```
%% transition function definition
fFun=@(x,u,w,k)[x(1)*x(2)+w(1);x(2)+w(2)];
f=nefHandleFunction(fFun,[2 0 2 0]);

%% measurement function definition
H=[1 0];
h=nefLinFunction(H,[],1);

%% description of random variables
pw=nefGaussianRV([0 0]',eye(2)*0.5);
pv=nefGaussianRV(0,0.01);
px0=nefGaussianRV([10;-0.85],1e-1*eye(2));

%% model definition and simulation
model=nefEqSystem(f,h,pw,pv,px0);
nSteps=20;
[z,x]=simulate(model,nSteps,[]);

%% estimator creation and use
UKF = nefUKF(model,'scalingParameter',0);
[estimates] = estimate(UKF,z,[]);
```

Figure: All components necessary for specification of the estimation experiment



Estimation experiment design for a probabilistically described system

Let the system dynamics be described by the following transitional pdf

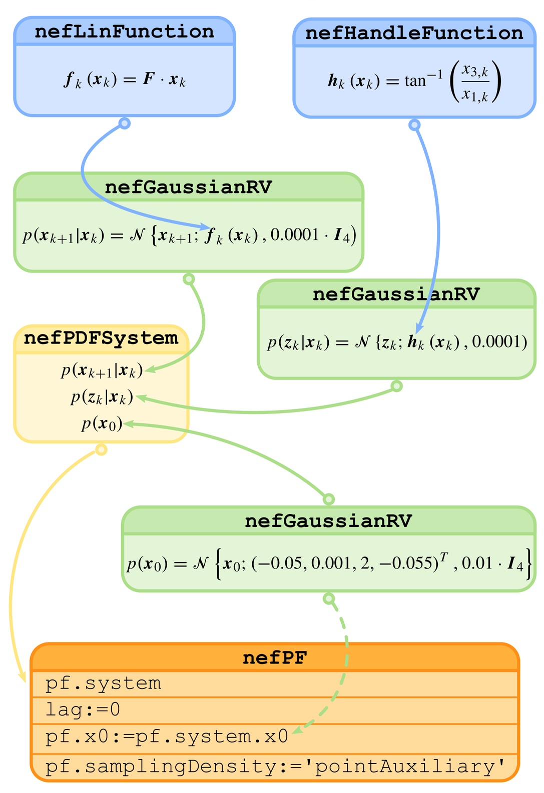
$$p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k) = \mathcal{N}\left\{\boldsymbol{x}_{k+1}; \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \boldsymbol{x}_k, 0.0001\boldsymbol{I}_4\right\},$$

where $\boldsymbol{I}_4$ is the 4-by-4 identity matrix. The measurement of the model is the bearing given by $\tan^{-1}(\frac{x_{3,k}}{x_{1,k}})$ and the corresponding measurement pdf is given as

$$p(z_k|\boldsymbol{x}_k) = \mathcal{N}\left\{z_k; \tan^{-1}(\frac{x_{3,k}}{x_{1,k}}), 0.0001\right\}.$$

```
%% transition pdf definition
F = [1 1 0 0;0 1 0 0 ;0 0 1 1 ;0 0 0 1];
xMean = nefLinFunction(F,[],[]);
xVariance = 0.0001*eye(4);
px = nefGaussianRV(xMean,xVariance);

%% measurement pdf definition
mFun = @(x,u,v,t) atan(x(3)/x(1));
zMean = nefHandleFunction(mFun,[4 0 0 0]);
zVariance = 0.0001;
pz = nefGaussianRV(zMean,zVariance);

%% prior state pdf definition
px0 = nefGaussianRV(...
        [-0.05 0.001 2 -0.055]',0.01*eye(4));

%% model definition and simulation
model = nefPDFSystem(px,pz,px0);
nSteps=20;
[z,x] = simulate(model,nSteps,[]);

%% estimator creation and use
PF = nefPF(model,...
        'samplingDensity','pointAuxiliary');
[estimates] = estimate(PF,z,[]);
```

Figure: All components necessary for specification of the estimation experiment



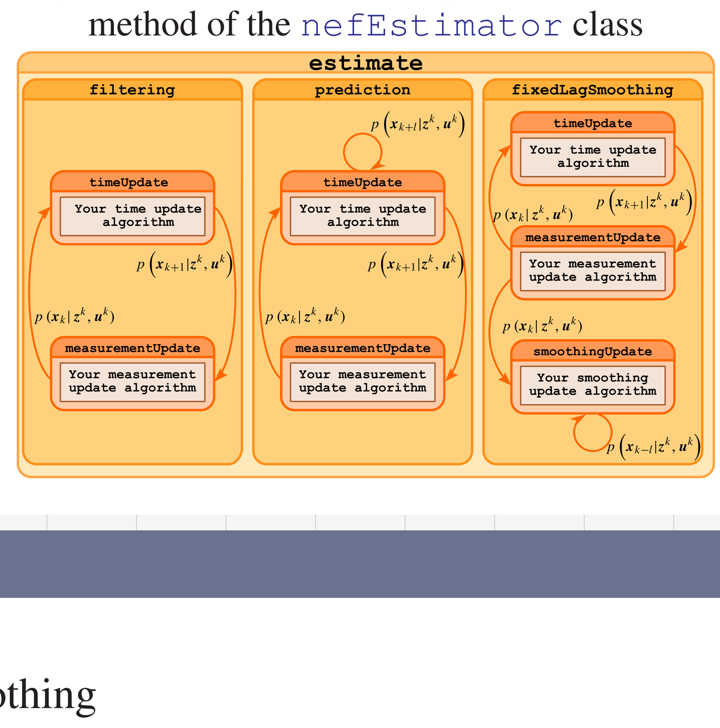
Implementation of user defined filters

For implementation of a new estimator it is sufficient to specify the essential algorithm for evaluation of filtering, prediction and smoothing pdf's only. The NEF handles the data storage and propagation.

Three steps to new estimator implementation

❶ create a child class the `nefEstimator` class
❷ implement the methods representing your algorithm (i.e. `measurementUpdate`, `timeUpdate` and `smoothUpdate`)
❸ implement new class constructor coping with optional parameters if necessary

Figure: Internal execution logic of `estimate` method of the `nefEstimator` class



Future plans

- implement additional estimation methods
- add support for fixed interval and fixed lag smoothing
- add support for multi-model problems
- provide support for advanced results visualisations (with camera ready output)
- provide GUI for user friendly preparation of complex estimation experiment setups