Nonlinear Estimation Software Framework in Optimal and Adaptive Control Problems

Miroslav Flídr, Ondřej Straka and Miroslav Šimandl

European Centre of Excellence ”NTIS - New Technologies for the Information Society”
Faculty of Applied Sciences
University of West Bohemia

Introduction to Nonlinear Estimation Framework (NEF)

The goal of the Nonlinear Estimation Framework (NEF)

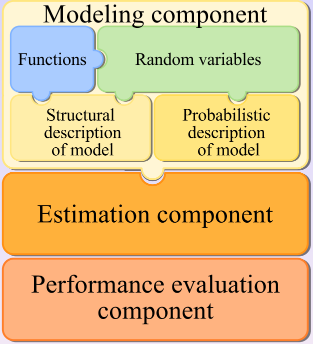
To provide a software framework designed for nonlinear state estimation of discrete-time stochastic dynamic systems.

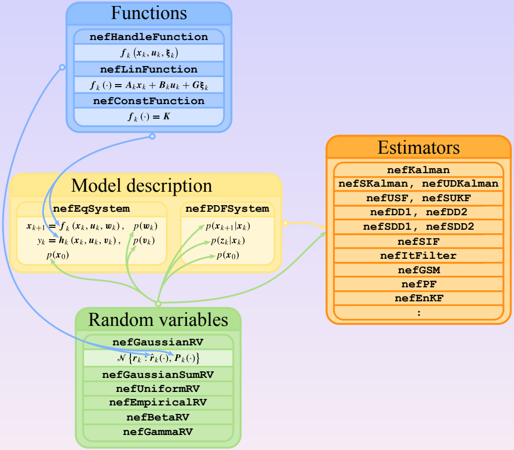
- a collection of MATLAB classes and functions for
  - modeling system behavior
  - state estimation
  - evaluation of the results
- development driven by the need for a tool that can
  - evaluate the quality of a state estimation method in arbitrary case
  - compare performance of several state estimators
  - provide means for effortless rapid prototyping of new state estimators
- can be easily incorporated into adaptive controller
  - provides state and parameters conditional probability density function

The main NEF features

- highly modular and extensible
- 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
- implements many of the popular nonlinear state estimators (both standard and **numerically stable** estimation algorithms),
- fast and easy estimation experiment setup
- facilitates implementation of **filtering**, **multi-step prediction** and **smoothing** tasks
- full estimator parametrization by means of the standard MATLAB property-value mechanism,
- evaluation of estimate quality.

NEF components and estimation experiment description





Scheme of NEF modeling component

Structural description

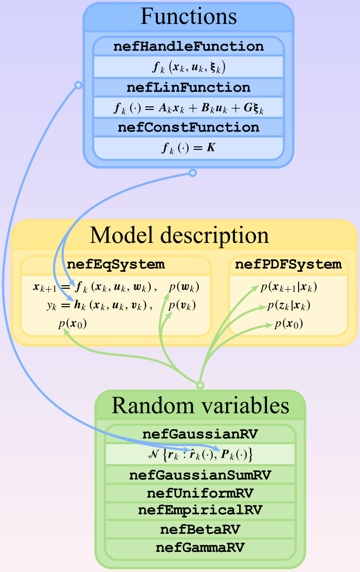
$$
\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k), \quad k = 0, 1, \ldots, \\
\boldsymbol{z}_k &= \boldsymbol{h}_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k), \quad k = 0, 1, \ldots,
\end{aligned}
$$

$$p(\boldsymbol{w}_k),\ p(\boldsymbol{v}_k),\ p(\boldsymbol{x}_0)$$

Probabilistic description

$$
\begin{aligned}
&p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k),\ k = 0, 1, \ldots, \\
&p(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{u}_k),\ k = 0, 1, \ldots, \\
&p(\boldsymbol{x}_0).
\end{aligned}
$$

$$\boldsymbol{x}_k \in \mathcal{R}^{n_x},\ \boldsymbol{z}_k \in \mathcal{R}^{n_z},\ \boldsymbol{u}_k \in \mathcal{R}^{n_u},$$
$$\boldsymbol{w}_k \in \mathcal{R}^{n_x},\ \boldsymbol{v}_k \in \mathcal{R}^{n_z}$$



Description of functions within NEF

Provided classes for description of multivariate functions	
`nefHandleFunction`	general function described by handle function
`nefLinFunction`	linear function
`nefConstFunction`	constant function
`nefFunction`	used by subclassing for complex function

`nefHandleFunction`

The most useful and common way of describing the functions.

Example: $h(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k, k) = \arctan\left(\frac{x_{2,k} - \sin(k)}{x_{1,k} - \cos(k)}\right) + v_k$

❶ create regular or handle function in MATLAB

```
mFun = @(x,u,v,k)...
          atan((x(2)-sin(k))/(x(1)-cos(k)))+v
```

❷ create `nefHandleFunction` instance with appropriate parameters

```
fun = nefHandleFunction(mFun,[2 0 1 1]);
```

Structurally and probabilistically described system

nefHandleFunction

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

nefGaussianRV

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

nefEqSystem

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{w}_k), \quad & p(\boldsymbol{w}_k) \\ y_k &= \boldsymbol{h}_k(\boldsymbol{x}_k, v_k), \quad & p(v_k) \\ & p(\boldsymbol{x}_0) \end{aligned}$$

nefLinFunction

$$\boldsymbol{h}_k(\boldsymbol{x}_k, v_k) = (1, 0)\, \boldsymbol{x}_k + v_k$$

nefGaussianRV

$$p(v_k) = \mathcal{N}\{v_k : 0, 0.01\}$$

nefGaussianRV

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

nefUKF

```
ukf.system
lag:=0
ukf.x0:=ukf.system.x0
```

nefLinFunction

$$\boldsymbol{f}_k(\boldsymbol{x}_k) = \boldsymbol{F} \cdot \boldsymbol{x}_k$$

nefHandleFunction

$$\boldsymbol{h}_k(\boldsymbol{x}_k) = \tan^{-1}\left(\frac{y_k}{x_k}\right)$$

nefGaussianRV

$$p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k) = \mathcal{N}\left\{\boldsymbol{x}_{k+1}; \boldsymbol{f}_k(\boldsymbol{x}_k), 0.0001 \cdot \boldsymbol{I}_4\right)$$

nefGaussianRV

$$p(z_k|\boldsymbol{x}_k) = \mathcal{N}\{z_k; \boldsymbol{h}_k(\boldsymbol{x}_k), 0.0001)$$

nefPDFSystem

$$p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k)$$
$$p(z_k|\boldsymbol{x}_k)$$
$$p(\boldsymbol{x}_0)$$

nefGaussianRV

$$p(\boldsymbol{x}_0) = \mathcal{N}\left\{\boldsymbol{x}_0; (-0.050.0012 - 0.055)^T, 0.01 \cdot \boldsymbol{I}_4\right)$$

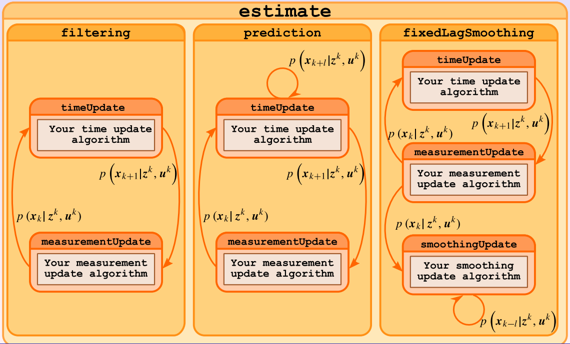
nefPF

```
pf.system
lag:=0
pf.x0:=pf.system.x0
pf.samplingDensity:='pointAuxiliary'
```

Estimation component

Estimation problem

- looking for the posterior pdf $p(\boldsymbol{x}_k|\boldsymbol{z}^\ell, \boldsymbol{u}^\ell)$
- solution provided by the Bayesian functional relations (BFR)
- mostly an approximate solution is being looked for
- BFR idea is embodied by class `nefEstimator`



Estimators implemented in the NEF estimation component

NEF class	estimators
`nefKalman`, `nefSKalman`, `nefUDKalman`	(extended) Kalman filter (standard, square-root and UD versions)
`nefDD1`, `nefSDD1`, `nefDD2`, `nefSDD2`	central difference Kalman filter, divided difference filter (1^{st} and 2^{nd} order) (standard and square-root version)
`nefUKF`, `nefSUKF`	unscented Kalman filter (standard and square-root version), cubature Kalman filter
`nefItKalman`	iterated Kalman filter
`nefGSM`	Gaussian sum filter
`nefPF`	bootstrap filter, generic particle filter, auxiliary particle filter, unscented particle filter
`nefEnKF`	ensemble Kalman filter
`nefSIF`	stochastic integration filter

Estimation tasks supported by individual estimators

estimator	filtering	prediction	smoothing
nefKalman	✓	✓	✓
nefSKalman	✓	✓	✓
nefUDKalman	✓	✓	
nefItKalman	✓	✓	✓
nefDD1	✓	✓	✓
nefSDD1	✓	✓	✓
nefDD2	✓	✓	✓
nefUKF	✓	✓	✓
nefSUKF	✓	✓	✓
nefGSM	✓	✓	
nefPF	✓	✓	
nefEnKF	✓	✓	
nefSIF	✓	✓	

Performance evaluation

Aim of this component

- to measure estimation error
- to compare performance of several estimators against the true value of the state

Steps to measure performance

1. collecting data from Monte Carlo simulations,
2. extracting appropriate indicators from the conditional distribution of the state provided by individual estimators
3. evaluating the performance index

Performance indices implemented in the NEF

ABSOLUTE ERROR MEASURES	
MSEM	mean squared error matrix
RMSE	root mean squared error
AEE	average Euclidean error
HAE	harmonic average error
GAE	geometric average error
MEDE	median error
MODE	mode error
RELATIVE ERROR MEASURES	
RMSRE	root mean squared relative error
ARE	average Euclidean relative error
BEEQ	Bayesian estimation error quotient
EMER	estimation error relative to measurement error
PERFORMANCE MEASURES	
NCI	non-credibility index
ANEES	average normalized estimation error squared

Simple control loop

❶ Initial condition

$p(\boldsymbol{x}_0|z^{-1})$

❶ Measurement update step

method `measurementUpdate` provides $p(\boldsymbol{x}_k|z^k)$

❷ Control law evaluation

custom code describing the evaluation of $\boldsymbol{u}_k$

❸ Time update step

method `timeUpdate` provides $p(\boldsymbol{x}_{k+1}|z^k, \boldsymbol{u}_k)$

Example of NEF use in control - problem statement

Considered system

$$
\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{w}_k, \\
z_k &= \boldsymbol{C}\boldsymbol{x}_k + v_k
\end{aligned}
$$

$$
\boldsymbol{A} = \begin{pmatrix} 0 & 1 \\ \theta_1 & \theta_2 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 0 \\ \theta_3 \end{pmatrix}, \boldsymbol{C} = (0\ 1)\,.
$$

$$
\boldsymbol{w}_k \sim \mathcal{N}\,(0, 0.0001)\,,\; v_k \sim \mathcal{N}\,(0, 0.001)\,.
$$

Criterion

$$
J = E\left\{\sum_{k=0}^{N-1} (\boldsymbol{x}_{k+1} - \bar{\boldsymbol{x}}_{k+1})^T \boldsymbol{Q}_{k+1}\,(\boldsymbol{x}_{k+1} - \bar{\boldsymbol{x}}_{k+1}) + \boldsymbol{u}_k^T \boldsymbol{R}_k \boldsymbol{u}_k\right\}
$$

$$
\boldsymbol{Q}_{k+1} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{R}_k = 0.001.
$$

LQG controller - problem description using NEF

Model description

```
f = nefLinFunction(A, B, eye(2));
h = nefLinFunction(C,[],1);
```

Initial condition

```
x0=[1;-0.5];
Px0=diag(0.2*[1 1]);
px0=nefGaussianRV(x0,Px0);
```

State and measurement noises

```
wmean=[0;0]; Pw=eye(2)*0.0001;
pw=nefGaussianRV(wmean,Pw);

vmean=0; Qv=0.001;
pv=nefGaussianRV(vmean,Pv);
```

Model

```
model=nefEqSystem(f,h,pw,pv,px0);
```

Estimator (UD Kalman filter)

```
UDKalman = nefUDKalman(model);
```

Note: In this case the parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$ are known!

LQG controller - implementation

❶ Initial condition

$$p(\boldsymbol{x}_0|\boldsymbol{z}^{-1}) = \mathcal{N}\left(\hat{\boldsymbol{x}}_0, P_0\right)$$

❶ Kalman filter - filtering step

$$\boldsymbol{\mu}_k(\boldsymbol{u}_0^{k-1}, \boldsymbol{y}_0^k) = \hat{\boldsymbol{x}}_k + \boldsymbol{K}_k^F(\boldsymbol{y}_k - \boldsymbol{C}_k\hat{\boldsymbol{x}}_k)$$

❷ LQ controller for CE system

$$\boldsymbol{u}_k(\boldsymbol{\mu}_k) = -\boldsymbol{K}_k\boldsymbol{\mu}_k + \boldsymbol{B}^T\left(\boldsymbol{F}_{k+1} - \boldsymbol{Q}_{k+1}\bar{\boldsymbol{x}}_{k+1}\right)$$

❸ Kalman filter - predictive step

$$\hat{\boldsymbol{x}}_{k+1}(\boldsymbol{u}^k{}_0, \boldsymbol{y}_0^k) = \boldsymbol{A}_k\boldsymbol{\mu}_k + \boldsymbol{B}_k\boldsymbol{u}_k$$

LQG controller - implementation

LQG control and trajectory simulation loop using NEF

```
predPDF.RV=nefGaussianRV(x0,nefUDFactorFunction(Px0));

[z(:,1),x(:,1),model] = simulate(model,1,[],'initialState',x0)

for k = 1:controlHorizon
  % determine current filtering pdf
  filtPDF = measurementUpdate(UDKalman,predPDF,[],z(:,k),k);
  mu(:,k) = evalMean(filtPDF.RV);

  % control law
  u(:,k)=-(B'*S{k+1}*B+R)\...
          (B'*S{k+1}*A*mu(:,k)+B'*F{k+1}-Q*xsetpoint(:,k+1));

  % determine one step predictive pdf
  predPDF = timeUpdate(UDKalman,filtPDF,u(:,k),k);

  % system trajectory simulation
  [z(:,time+1),x(:,time+1),model] = simulate(model,1,u(:,time));
end
```

Cautious controller - setup modifications

Note: In this case the parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$ are unknown!

- the system is bi-linear form estimation point of view

Modified state dynamic description for estimation

```
fFun = @(x,u,w,k) [x(2)+w(1);
                    x(3)*x(1)+x(4)*x(2)+x(5)*u+w(2);
                    x(3); x(4); x(5)];
f = nefHandleFunction(fFun,[5 1 2 0]);
```

- it is possible to use `nefUDKalman` again
  - the UD factorized extended Kalman filter would be used
  - it requires specification of first derivative in `nefHandleFunction` constructor
- the `nefUKF` is better choice

```
UKF = nefUKF(model);
```

- control law given as $\boldsymbol{u}_k = \operatorname{argmin}_{\boldsymbol{u}_k} \mathcal{L}_k(\boldsymbol{x}_k, \boldsymbol{u}_k)$

Cautious controller - implementation

2/2

Cautious control and trajectory simulation loop using NEF

```
[z(:,1),x(:,1),model] = simulate(model,1,[],'initialState',x0)
for k = 1:controlHorizon
  % determine current filtering pdf
  filtPDF = measurementUpdate(UKF,predPDF,[],z(:,k),k);

  % extract mean and covariance matrix
  mu(:,k) = evalMean(filtPDF);
  Pf = evalVariance(filtPDF);

  % construct estimated matrices A and B
  estA = [ 0 1; mu(3:4,k)'];
  estB = [ 0; mu(5,k)];

  % cautious control law
  u(:,k)=-(estB'*Q*estB+Pf(5,5)+R)\...
          (estB'*(Q*estA+Pf(3:4,3:4)*mu(1:2,k)-estB'*Q*xsetpoint(:,k+1));

  % determine one step predictive pdf
  predPDF = timeUpdate(UDKalman,...
  filtPDF,u(:,k),k);

  % system trajectory simulation
  [z(:,time+1),x(:,time+1),model] = simulate(model,1,u(:,time));
end
```

Concluding remarks

Recapitulation

- ✔ versatile tool for testing estimators
- ✔ can be easily incorporated into adaptive control law
- ✔ problem specification maximally simplified
- ✔ provides means even for complex model description
- ✔ offers various performance indexes
- ✔ possible rapid prototyping of user defined estimators

Additional information

- ➩ free for non-commercial use
- ➩ current stable version accessible at **`http://nft.kky.zcu.cz/`**