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
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

Modeling component
- Functions
  - Model description
    - Structural description of model
    - Probabilistic description of model
  - Random variables

Estimation component
- Estimators
  - nefKalman, nefSKalman, nefUDKalman
  - nefUSF, nefSUKF
  - nefDD1, nefDD2
  - nefSDD1, nefSDD2
  - nefSIF
  - nefItFilter
  - nefGSM
  - nefFF
  - nefEnKF

Performance evaluation component
- Random variables
  - nefGaussianRV
  - \( N \{ \tilde{f}_k(x_k), P_k \} \)
  - nefGaussianSumRV
  - nefUniformRV
  - nefEmpiricalRV
  - nefBetaRV
  - nefGammaRV

Model description
- nefEqSystem
  - \( x_{k+1} = f_k(x_k, u_k, w_k), \quad p(w_k) \)
  - \( y_k = h_k(x_k, u_k, v_k), \quad p(v_k) \)
  - \( p(x_0) \)
- nefPDFSystem
  - \( p(x_{k+1}|x_k) \)
  - \( p(z_k|x_k) \)
  - \( p(x_k) \)

Functions
- nefHandleFunction
- nefLinFunction
  - \( f_k() = A_k x_k + B_k u_k + G_k \)
- nefConstFunction
  - \( f_k() = K \)
Scheme of NEF modeling component

Structural description

\[ x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \ldots, \]
\[ z_k = h_k(x_k, u_k, v_k), \quad k = 0, 1, \ldots, \]
\[ p(w_k), p(v_k), p(x_0) \]

Probabilistic description

\[ p(x_{k+1}|x_k, u_k), \quad k = 0, 1, \ldots, \]
\[ p(z_k|x_k, u_k), \quad k = 0, 1, \ldots, \]
\[ p(x_0). \]

Functions

- nefHandleFunction:
  \[ f_k(x_k, u_k, \xi_k) \]
- nefLinFunction:
  \[ f_k() = A_kx_k + B_ku_k + G\xi_k \]
- nefConstFunction:
  \[ f_k() = K \]

Model description

- nefEqSystem:
  \[ x_{k+1} = f_k(x_k, u_k, w_k), \quad p(w_k) \]
  \[ y_k = h_k(x_k, u_k, v_k), \quad p(v_k) \]
  \[ p(x_0) \]
- nefPDFSystem:
  \[ p(x_{k+1}|x_k) \]
  \[ p(z_k|x_k) \]
  \[ p(x_0) \]

Random variables

- nefGaussianRV
  \[ \mathcal{N}\{r_k; \mu_k(), \sigma^2_k()\} \]
- nefGaussianSumRV
- nefUniformRV
- nefEmpiricalRV
- nefBetaRV
- nefGammaRV
Description of functions within NEF

### Provided classes for description of multivariate functions

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nefHandleFunction</td>
<td>general function described by handle function</td>
</tr>
<tr>
<td>nefLinFunction</td>
<td>linear function</td>
</tr>
<tr>
<td>nefConstFunction</td>
<td>constant function</td>
</tr>
<tr>
<td>nefFunction</td>
<td>used by subclassing for complex function</td>
</tr>
</tbody>
</table>

**nefHandleFunction**

The most useful and common way of describing the functions.

Example: \( h(x_k, u_k, v_k, k) = \arctan\left(\frac{x_{2,k} - \sin(k)}{x_{1,k} - \cos(k)}\right) + v_k \)

1. create regular or handle function in MATLAB
   ```matlab
   mFun = @(x,u,v,k)... 
   \text{atan((x(2)-\sin(k))/(x(1)-\cos(k))))+v}
   ```

2. create **nefHandleFunction** instance with appropriate parameters
   ```matlab
   fun = nefHandleFunction(mFun,[2 0 1 1]);
   ```
Structurally and probabilistically described system

\[ f_k(x_k, w_k) = \left( \frac{x_{2,k} \cdot x_{1,k}}{x_{2,k}} \right) + w_k \]

\[ p(w_k) = \mathcal{N}\left(w_k : \begin{pmatrix} 0 \\ 0 \\ 0.5 \\ 0 \\ 0.5 \end{pmatrix} \right) \]

\[ x_{k+1} = f_k(x_k, w_k), \quad p(w_k) \]
\[ y_k = h_k(x_k, v_k), \quad p(v_k) \]
\[ p(x_0) \]

\[ h_k(x_k, v_k) = (1, 0)x_k + v_k \]

\[ p(v_k) = \mathcal{N}\left(v_k : 0.01\right) \]

\[ p(x_0) = \mathcal{N}\left(x_0 : \begin{pmatrix} 10 \\ -0.85 \\ 0.1 \\ 0 \\ 0.1 \end{pmatrix} \right) \]

\[ pf.system \]
\[ lag:=0 \]
\[ pf.x0:=pf.system.x0 \]
\[ pf.samplingDensity:='pointAuxiliary' \]
Estimation component

Estimation problem

- looking for the posterior pdf $p(x_k | z^\ell, u^\ell)$
- 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

<table>
<thead>
<tr>
<th>NEF class</th>
<th>estimators</th>
</tr>
</thead>
<tbody>
<tr>
<td>nefKalman, nefSKalman,</td>
<td>(extended) Kalman filter (standard, square-root and UD versions)</td>
</tr>
<tr>
<td>nefUDKalman</td>
<td></td>
</tr>
<tr>
<td>nefDD1, nefSDD1, nefDD2</td>
<td>central difference Kalman filter, divided difference filter (1&lt;sup&gt;st&lt;/sup&gt; and 2&lt;sup&gt;nd&lt;/sup&gt; order) (standard and square-root version)</td>
</tr>
<tr>
<td>nefSDD2</td>
<td></td>
</tr>
<tr>
<td>nefUKF, nefSUKF</td>
<td>unscented Kalman filter (standard and square-root version), cubature Kalman filter</td>
</tr>
<tr>
<td>nefItKalman</td>
<td>iterated Kalman filter</td>
</tr>
<tr>
<td>nefGSM</td>
<td>Gaussian sum filter</td>
</tr>
<tr>
<td>nefPF</td>
<td>bootstrap filter, generic particle filter, auxiliary particle filter, unscented particle filter</td>
</tr>
<tr>
<td>nefEnKF</td>
<td>ensemble Kalman filter</td>
</tr>
<tr>
<td>nefSIF</td>
<td>stochastic integration filter</td>
</tr>
</tbody>
</table>
### Estimation tasks supported by individual estimators

<table>
<thead>
<tr>
<th>estimator</th>
<th>filtering</th>
<th>prediction</th>
<th>smoothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>nefKalman</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nefSKalman</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nefUDKalman</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>nefItKalman</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nefDD1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nefSDD1</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>nefDD2</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>nefUKF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nefSUKF</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>nefGSM</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>nefPF</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>nefEnKF</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>nefSIF</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>ABSOLUTE ERROR MEASURES</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MSEM</td>
<td>mean squared error matrix</td>
</tr>
<tr>
<td>RMSE</td>
<td>root mean squared error</td>
</tr>
<tr>
<td>AEE</td>
<td>average Euclidean error</td>
</tr>
<tr>
<td>HAE</td>
<td>harmonic average error</td>
</tr>
<tr>
<td>GAE</td>
<td>geometric average error</td>
</tr>
<tr>
<td>MEDE</td>
<td>median error</td>
</tr>
<tr>
<td>MODE</td>
<td>mode error</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RELATIVE ERROR MEASURES</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSRE</td>
<td>root mean squared relative error</td>
</tr>
<tr>
<td>ARE</td>
<td>average Euclidean relative error</td>
</tr>
<tr>
<td>BEEQ</td>
<td>Bayesian estimation error quotient</td>
</tr>
<tr>
<td>EMER</td>
<td>estimation error relative to measurement error</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PERFORMANCE MEASURES</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NCI</td>
<td>non-credibility index</td>
</tr>
<tr>
<td>ANEES</td>
<td>average normalized estimation error squared</td>
</tr>
</tbody>
</table>
**Simple control loop**

1. **Initial condition**
   
   \[ p(x_0|z^{-1}) \]

2. **Measurement update step**
   
   Method `measurementUpdate` provides \( p(x_k|z^k) \)

3. **Control law evaluation**
   
   Custom code describing the evaluation of \( u_k \)

4. **Time update step**
   
   Method `timeUpdate` provides \( p(x_{k+1}|z^k, u_k) \)
Example of NEF use in control - problem statement

Considered system

\[ x_{k+1} = Ax_k + Bu_k + w_k, \]
\[ z_k = Cx_k + v_k \]

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

\[ w_k \sim \mathcal{N}(0, 0.0001), \ v_k \sim \mathcal{N}(0, 0.001). \]

Criterion

\[ J = E \left\{ \sum_{k=0}^{N-1} (x_{k+1} - \bar{x}_{k+1})^T Q_{k+1} (x_{k+1} - \bar{x}_{k+1}) + u_k^T R_k u_k \right\} \]

\[ Q_{k+1} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \quad R_k = 0.001. \]
LQG controller - problem description using NEF

Model description
\[
f = \text{nefLinFunction}(A, B, \text{eye}(2)); \\
h = \text{nefLinFunction}(C, [], 1);
\]

Initial condition
\[
x0 = [1; -0.5]; \\
Px0 = \text{diag}(0.2 * [1 1]); \\
px0 = \text{nefGaussianRV}(x0, Px0);
\]

State and measurement noises
\[
wmean = [0; 0]; \\
Pw = \text{eye}(2) * 0.0001; \\
pw = \text{nefGaussianRV}(wmean, Pw);
\]
\[
vmean = 0; \\
Qv = 0.001; \\
pv = \text{nefGaussianRV}(vmean, Pv);
\]

Model
\[
\text{model} = \text{nefEqSystem}(f, h, pw, pv, px0);
\]

Estimator (UD Kalman filter)
\[
\text{UDKalman} = \text{nefUDKalman(model)};
\]

Note: In this case the parameters \( \theta = (\theta_1, \theta_2, \theta_3) \) are known!
LQG controller - implementation

1. Initial condition
\[ p(x_0|z^{-1}) = \mathcal{N}(\hat{x}_0, P_0) \]

2. Kalman filter - filtering step
\[ \mu_k(u^{k-1}_0, y^k_0) = \hat{x}_k + K_k^F (y_k - C_k \hat{x}_k) \]

3. LQ controller for CE system
\[ u_k(\mu_k) = -K_k \mu_k + B^T (F_{k+1} - Q_{k+1} \tilde{x}_{k+1}) \]

4. Kalman filter - predictive step
\[ \hat{x}_{k+1}(u^k_0, y^k_0) = A_k \mu_k + B_k u_k \]
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)
    &lt;-(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, l, u(:,time));
end
Cautious controller - setup modifications

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

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

Modified state dynamic description for estimation

```matlab
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

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

- control law given as $u_k = \arg\min_{u_k} \mathcal{L}_k (x_k, u_k)$
Cautious controller - implementation

```matlab
[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