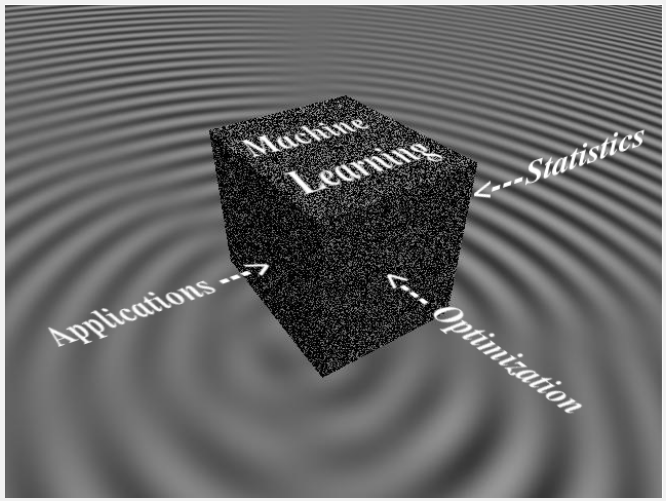
Identification of Monotone Wiener Systems

Kristiaan Pelckmans

UU/IT, SysCon (Liang Dai) + ESAT/KULeuven/Belgium (Van Belle, Suykens, Van Huffel)

Lund 2011, 5th Swedish-Chinese Conference on Control



Overview

1. Problem Setting - Monotone Wiener Models.
2. Batch Identification - MINLIP.
3. Recursive Identification - RANKTRON.
4. Non-parameteric Control - NORTKNAR.
5. Complexity Control.
6. Conclusions and Outlook.

$\rightarrow$ Specific 2 General.

Overview

1. Problem Setting - Monotone Wiener Models.
2. Batch Identification - MINLIP.
3. Recursive Identification - RANKTRON.
4. Non-parameteric Control - NORTKNAR.
5. Complexity Control.
6. Conclusions and Outlook.

$\rightarrow$ Specific 2 General.

Overview

1. Problem Setting - Monotone Wiener Models.
2. Batch Identification - MINLIP.
3. Recursive Identification - RANKTRON.
4. Non-parameteric Control - NORTKNAR.
5. Complexity Control.
6. Conclusions and Outlook.

$\rightarrow$ Specific 2 General.

Overview

1. Problem Setting - Monotone Wiener Models.
2. Batch Identification - MINLIP.
3. Recursive Identification - RANKTRON.
4. Non-parameteric Control - NORTKNAR.
5. Complexity Control.
6. Conclusions and Outlook.

$\rightarrow$ Specific 2 General.

Overview

1. Problem Setting - Monotone Wiener Models.
2. Batch Identification - MINLIP.
3. Recursive Identification - RANKTRON.
4. Non-parameteric Control - NORTKNAR.
5. Complexity Control.
6. Conclusions and Outlook.

$\rightarrow$ Specific 2 General.

Overview

1. Problem Setting - Monotone Wiener Models.
2. Batch Identification - MINLIP.
3. Recursive Identification - RANKTRON.
4. Non-parameteric Control - NORTKNAR.
5. Complexity Control.
6. Conclusions and Outlook.

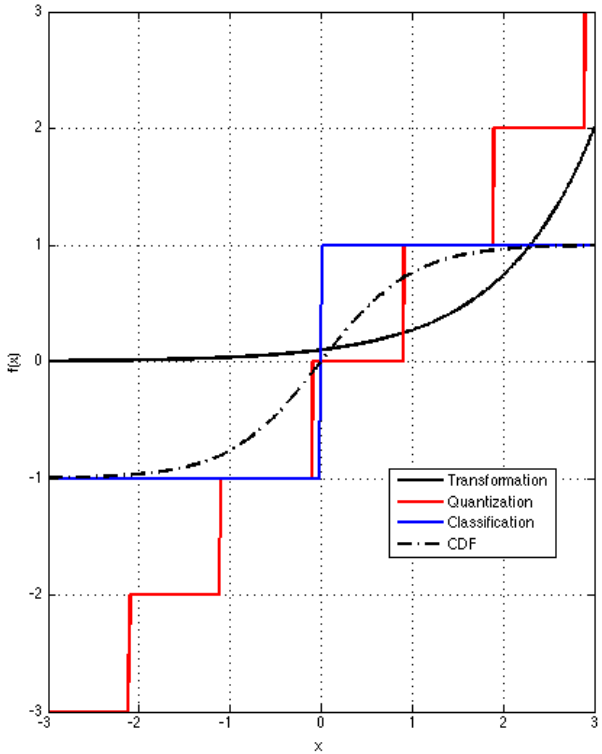
$\rightarrow$ Specific 2 General.

Overview

1. Problem Setting - Monotone Wiener Models.
2. Batch Identification - MINLIP.
3. Recursive Identification - RANKTRON.
4. Non-parameteric Control - NORTKNAR.
5. Complexity Control.
6. Conclusions and Outlook.

$\rightarrow$ Specific 2 General.

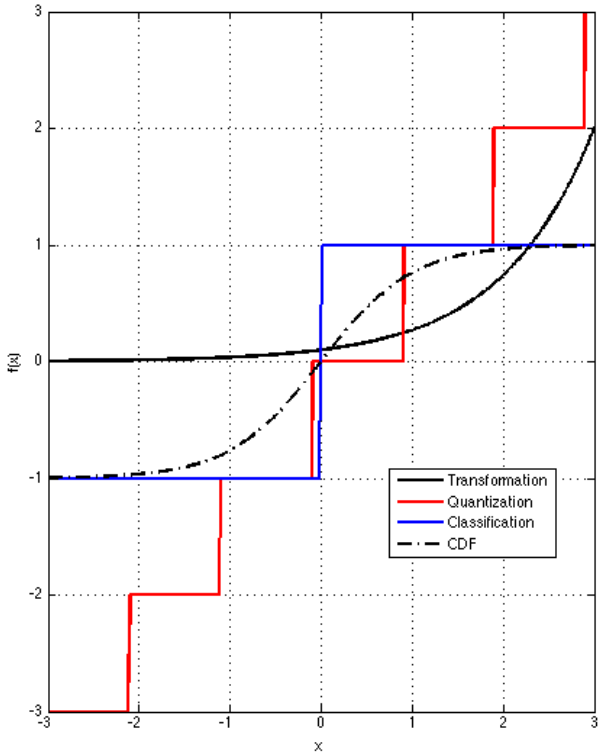
Problem Setting



$$y_t = f_0\left(H_0(q^{-1})u_t\right) = f_0\left(\sum_{\tau=0}^{\infty} h_{0,\tau}\, u_{t-\delta}\right),\ \forall t = 1..n$$

- Identification: given $\{(u_t, y_t)\}_{t=1}^{n}$, recover unknown but fixed $(f_0, \mathbf{h}_0)$.
- Monotone: f_0 is monotone in/decreasing.
  - Saturation, e.g.
    $$y_t = f_0(z_t) = \min(z_t, 1).$$
  - Transformed, e.g.
    $$y_t = f_0(z_t) = \log z_t.$$
  - Quantized (set-valued), e.g.
    $$y_t = f_0(z_t) = \lfloor z_t \rfloor.$$
  - Deadzones, e.g.
    $$y_t = f_0(z_t) = I(|z_t| > 1) z_t.$$

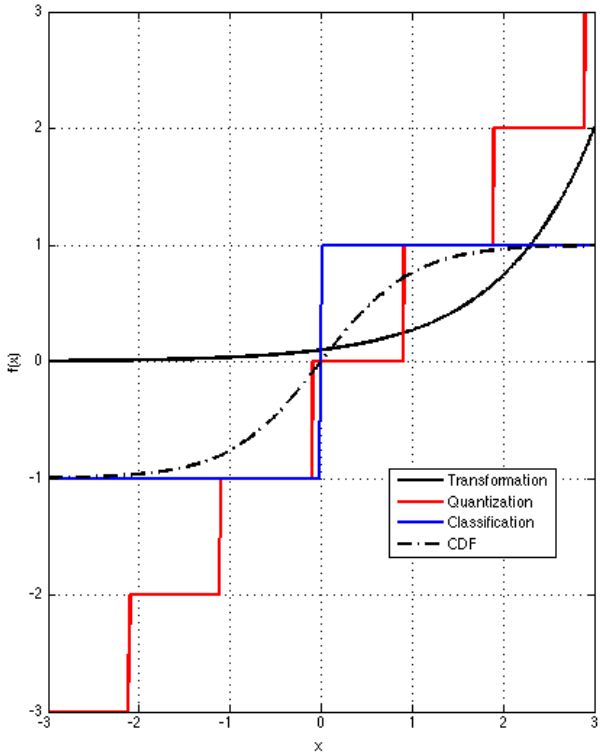
Problem Setting



$$y_t = f_0\left(H_0(q^{-1})u_t\right) = f_0\left(\sum_{\tau=0}^{\infty} h_{0,\tau}\, u_{t-\delta}\right),\ \forall t = 1..n$$

- Identification: given $\{(u_t, y_t)\}_{t=1}^{n}$, recover unknown but fixed $(f_0, \mathbf{h}_0)$.
- Monotone: f_0 is monotone in/decreasing.
  - Saturation, e.g.
    $$y_t = f_0(z_t) = \min(z_t, 1).$$
  - Transformed, e.g.
    $$y_t = f_0(z_t) = \log z_t.$$
  - Quantized (set-valued), e.g.
    $$y_t = f_0(z_t) = \lfloor z_t \rfloor.$$
  - Deadzones, e.g.
    $$y_t = f_0(z_t) = I(|z_t| > 1) z_t.$$

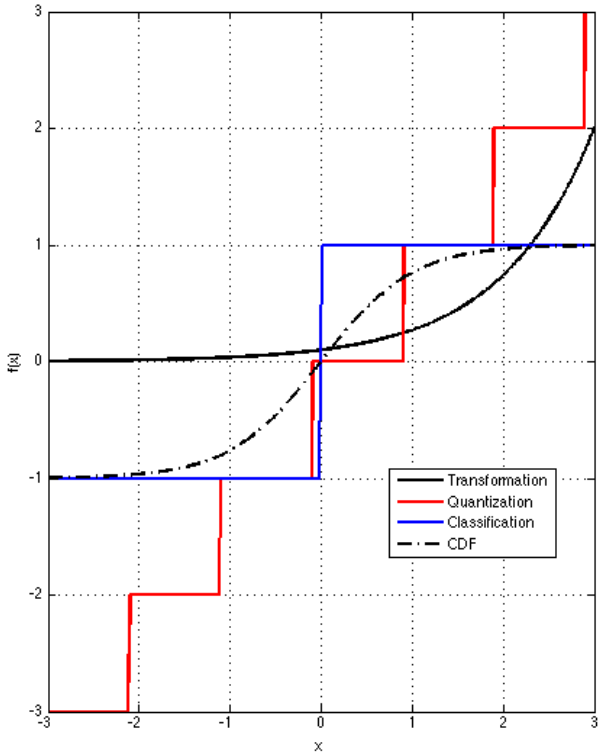
Problem Setting



$$y_t = f_0\left(H_0(q^{-1})u_t\right) = f_0\left(\sum_{\tau=0}^{\infty} h_{0,\tau}\, u_{t-\delta}\right),\ \forall t = 1..n$$

- Identification: given $\{(u_t, y_t)\}_{t=1}^{n}$, recover unknown but fixed $(f_0, \mathbf{h}_0)$.
- Monotone: f_0 is monotone in/decreasing.
  - Saturation, e.g.
    $$y_t = f_0(z_t) = \min(z_t, 1).$$
  - Transformed, e.g.
    $$y_t = f_0(z_t) = \log z_t.$$
  - Quantized (set-valued), e.g.
    $$y_t = f_0(z_t) = \lfloor z_t \rfloor.$$
  - Deadzones, e.g.
    $$y_t = f_0(z_t) = I(|z_t| > 1) z_t.$$

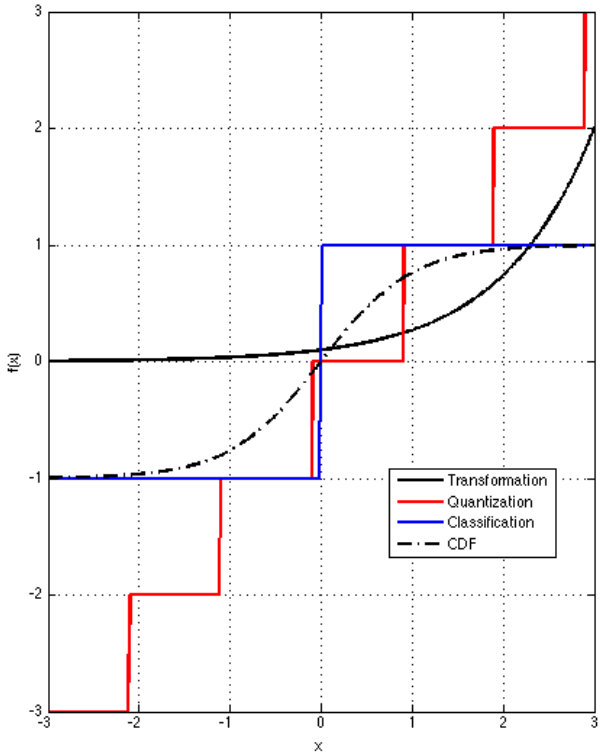
Problem Setting



$$y_t = f_0\left(H_0(q^{-1})u_t\right) = f_0\left(\sum_{\tau=0}^{\infty} h_{0,\tau}\, u_{t-\delta}\right),\ \forall t = 1..n$$

- Identification: given $\{(u_t, y_t)\}_{t=1}^n$, recover unknown but fixed $(f_0, \mathbf{h}_0)$.
- Monotone: f_0 is monotone in/decreasing.
  - Saturation, e.g.
    $$y_t = f_0(z_t) = \min(z_t, 1).$$
  - Transformed, e.g.
    $$y_t = f_0(z_t) = \log z_t.$$
  - Quantized (set-valued), e.g.
    $$y_t = f_0(z_t) = \lfloor z_t \rfloor.$$
  - Deadzones, e.g.
    $$y_t = f_0(z_t) = I(|z_t| > 1) z_t.$$

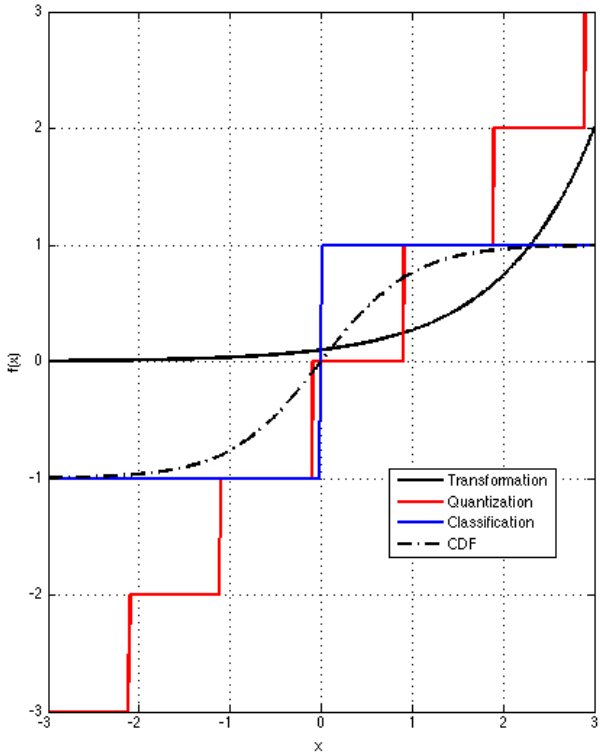
Problem Setting



$$y_t = f_0\left(H_0(q^{-1})u_t\right) = f_0\left(\sum_{\tau=0}^{\infty} h_{0,\tau}\, u_{t-\delta}\right),\ \forall t = 1..n$$

- Identification: given $\{(u_t, y_t)\}_{t=1}^n$, recover unknown but fixed $(f_0, \mathbf{h}_0)$.
- Monotone: f_0 is monotone in/decreasing.
  - Saturation, e.g.
    $$y_t = f_0(z_t) = \min(z_t, 1).$$
  - Transformed, e.g.
    $$y_t = f_0(z_t) = \log z_t.$$
  - Quantized (set-valued), e.g.
    $$y_t = f_0(z_t) = \lfloor z_t \rfloor.$$
  - Deadzones, e.g.
    $$y_t = f_0(z_t) = I(|z_t| > 1) z_t.$$

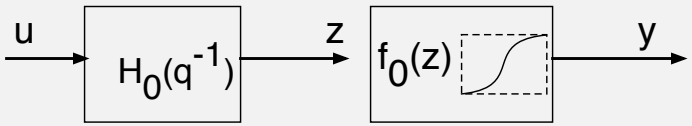
Problem Setting



$$y_t = f_0\left(H_0(q^{-1})u_t\right) = f_0\left(\sum_{\tau=0}^{\infty} h_{0,\tau}\, u_{t-\delta}\right),\ \forall t = 1..n$$

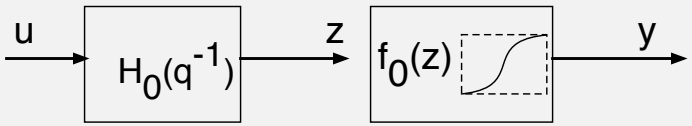
- Identification: given $\{(u_t, y_t)\}_{t=1}^{n}$, recover unknown but fixed $(f_0, \mathbf{h}_0)$.
- Monotone: f_0 is monotone in/decreasing.
  - Saturation, e.g.
    $$y_t = f_0(z_t) = \min(z_t, 1).$$
  - Transformed, e.g.
    $$y_t = f_0(z_t) = \log z_t.$$
  - Quantized (set-valued), e.g.
    $$y_t = f_0(z_t) = \lfloor z_t \rfloor.$$
  - Deadzones, e.g.
    $$y_t = f_0(z_t) = I(|z_t| > 1) z_t.$$

Monotone Wiener Systems



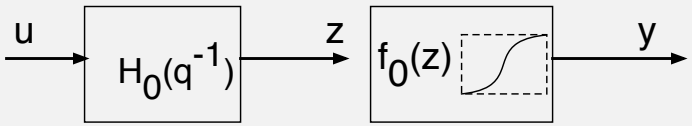
- Noise
- Identifiability.
- Linear-Nonlinear.
- No stochastic assumptions, non-differentiable, discontinuous functions, ...

Monotone Wiener Systems



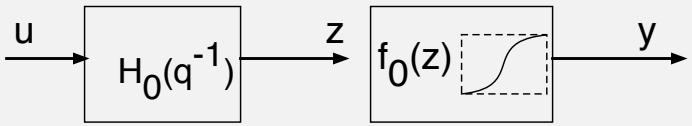
- Noise
- Identifiability.
- Linear-Nonlinear.
- No stochastic assumptions, non-differentiable, discontinuous functions, ...

Monotone Wiener Systems



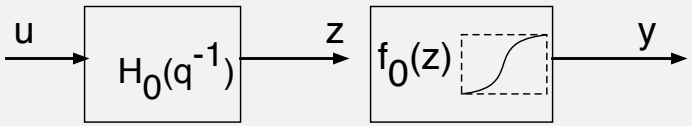
- Noise
- Identifiability.
- Linear-Nonlinear.
- No stochastic assumptions, non-differentiable, discontinuous functions, ...

Monotone Wiener Systems



- Noise
- Identifiability.
- Linear-Nonlinear.
- No stochastic assumptions, non-differentiable, discontinuous functions, ...

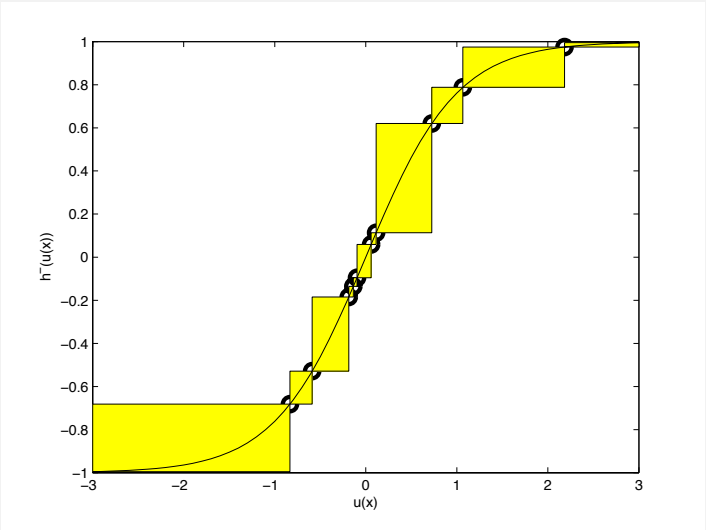
Monotone Wiener Systems



- Noise
- Identifiability.
- Linear-Nonlinear.
- No stochastic assumptions, non-differentiable, discontinuous functions, ...

Batch Setting - MINLIP

$$\arg\min_{\mathbf{h}} \|\mathbf{h}\|_2 \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

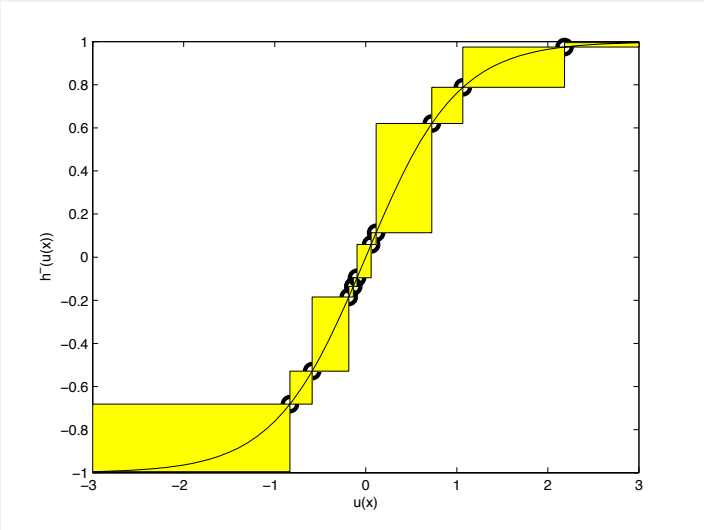


- Preserve order ('monotone')
- Least Complex.
- If too simple a solution $\Rightarrow$ counterexample.
- Convex.
- Cfr.:
  - Regularization - two equally powerful solutions $\rightarrow$ numerical problems.
  - ill-posed problems.
  - Bayesian Prior.
  - SVM: Least Complex without classification error.

Automatica 2011, CDC 2010, ...

Batch Setting - MINLIP

$$\arg\min_{\mathbf{h}} \|\mathbf{h}\|_2 \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

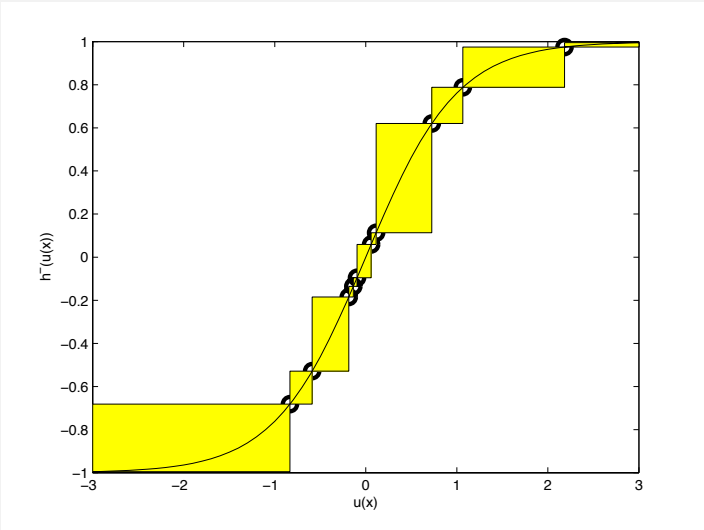


- Preserve order ('monotone')
- Least Complex.
- If too simple a solution $\Rightarrow$ counterexample.
- Convex.
- Cfr.:
  - Regularization - two equally powerful solutions $\rightarrow$ numerical problems.
  - ill-posed problems.
  - Bayesian Prior.
  - SVM: Least Complex without classification error.

Automatica 2011, CDC 2010, ...

Batch Setting - MINLIP

$$\arg\min_{\mathbf{h}} \|\mathbf{h}\|_2 \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

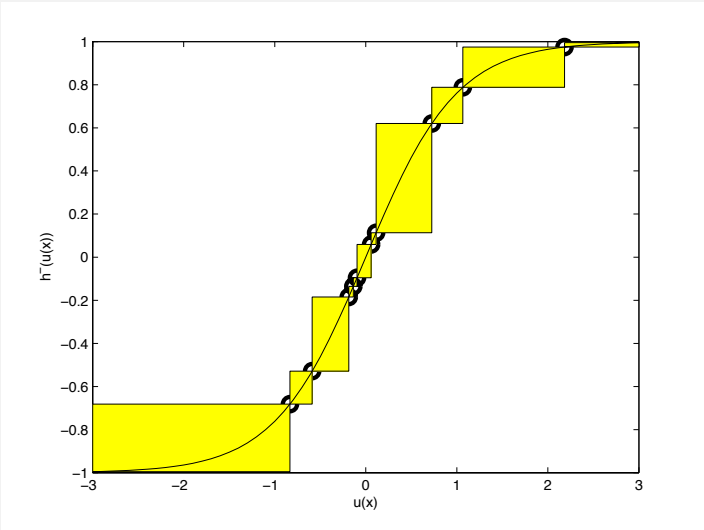


- Preserve order ('monotone')
- Least Complex.
- If too simple a solution $\Rightarrow$ counterexample.
- Convex.
- Cfr.:
  - Regularization - two equally powerful solutions $\rightarrow$ numerical problems.
  - ill-posed problems.
  - Bayesian Prior.
  - SVM: Least Complex without classification error.

Automatica 2011, CDC 2010, ...

Batch Setting - MINLIP

$$\arg\min_{\mathbf{h}} \|\mathbf{h}\|_2 \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

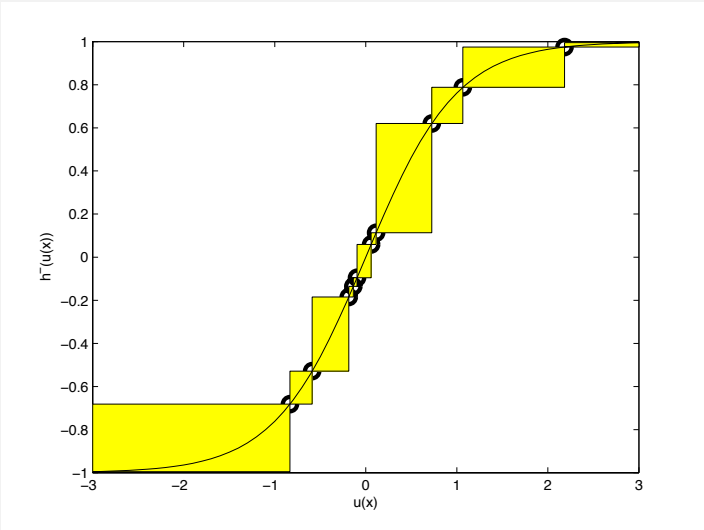


- Preserve order ('monotone')
- Least Complex.
- If too simple a solution $\Rightarrow$ counterexample.
- Convex.
- Cfr.:
  - Regularization - two equally powerful solutions $\rightarrow$ numerical problems.
  - ill-posed problems.
  - Bayesian Prior.
  - SVM: Least Complex without classification error.

Automatica 2011, CDC 2010, ...

Batch Setting - MINLIP

$$\arg\min_{\mathbf{h}} \|\mathbf{h}\|_2 \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

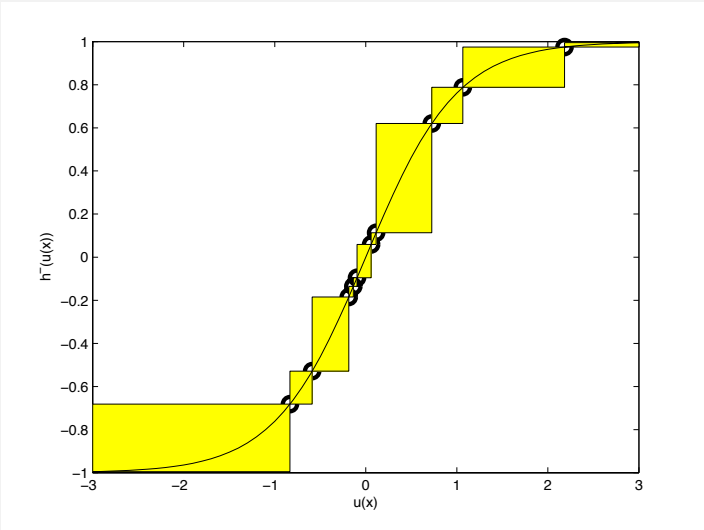


- Preserve order ('monotone')
- Least Complex.
- If too simple a solution $\Rightarrow$ counterexample.
- Convex.
- Cfr.:
  - Regularization - two equally powerful solutions $\rightarrow$ numerical problems.
  - ill-posed problems.
  - Bayesian Prior.
  - SVM: Least Complex without classification error.

Automatica 2011, CDC 2010, ...

Batch Setting - MINLIP

$$\arg\min_{\mathbf{h}} \|\mathbf{h}\|_2 \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

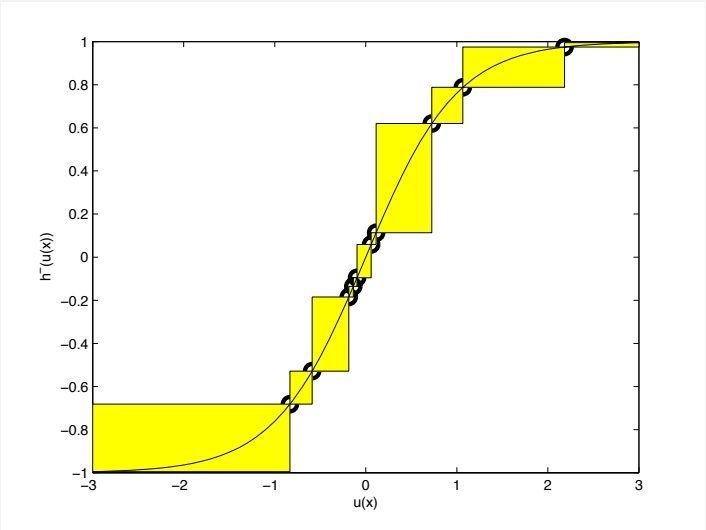


- Preserve order ('monotone')
- Least Complex.
- If too simple a solution $\Rightarrow$ counterexample.
- Convex.
- Cfr.:
  - Regularization - two equally powerful solutions $\rightarrow$ numerical problems.
  - ill-posed problems.
  - Bayesian Prior.
  - SVM: Least Complex without classification error.

Automatica 2011, CDC 2010, ...

Batch Setting - MINLIP

$$\arg\min_{\mathbf{h}} \|\mathbf{h}\|_2 \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

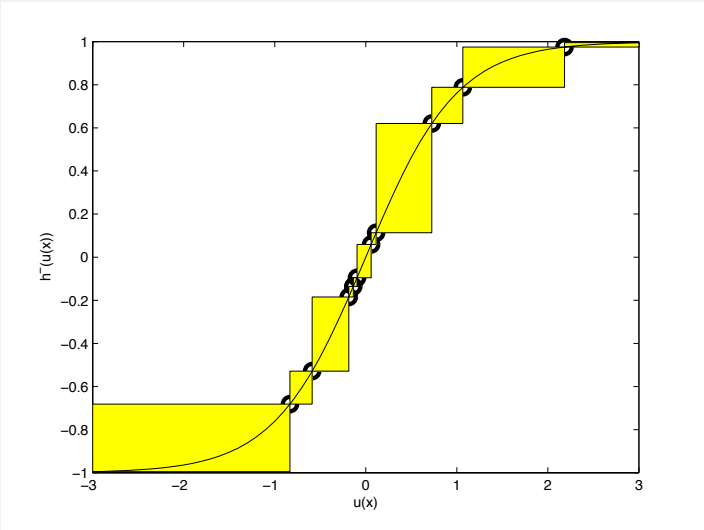


- Preserve order ('monotone')
- Least Complex.
- If too simple a solution $\Rightarrow$ counterexample.
- Convex.
- Cfr.:
  - Regularization - two equally powerful solutions $\rightarrow$ numerical problems.
  - ill-posed problems.
  - Bayesian Prior.
  - SVM: Least Complex without classification error.

Automatica 2011, CDC 2010, ...

Batch Setting - MINLIP

$$\arg\min_{\mathbf{h}} \|\mathbf{h}\|_2 \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

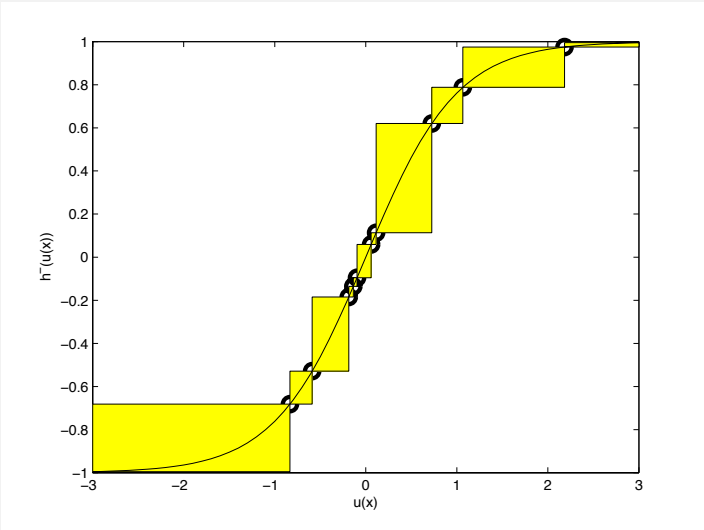


- Preserve order ('monotone')
- Least Complex.
- If too simple a solution $\Rightarrow$ counterexample.
- Convex.
- Cfr.:
  - Regularization - two equally powerful solutions $\rightarrow$ numerical problems.
  - ill-posed problems.
  - Bayesian Prior.
  - SVM: Least Complex without classification error.

Automatica 2011, CDC 2010, ...

Batch Setting - MINLIP

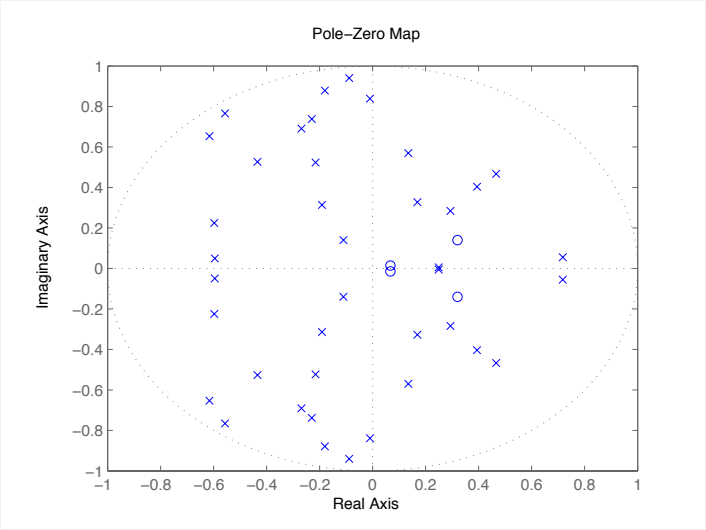
$$\arg\min_{\mathbf{h}} \|\mathbf{h}\|_2 \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

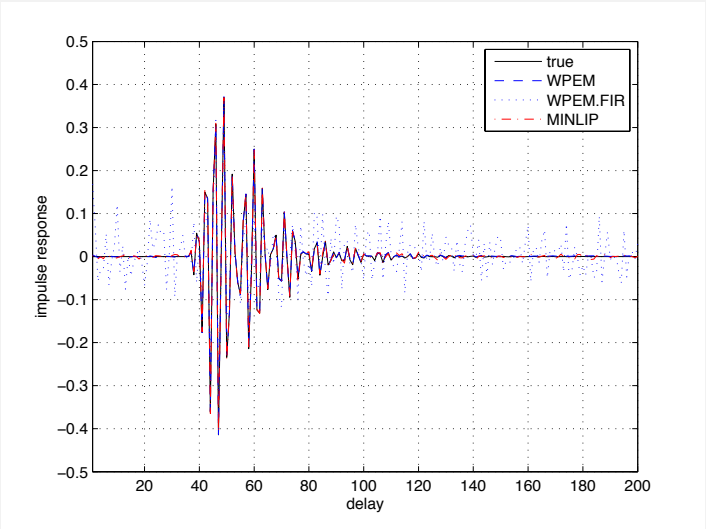


- Preserve order ('monotone')
- Least Complex.
- If too simple a solution $\Rightarrow$ counterexample.
- Convex.
- Cfr.:
  - Regularization - two equally powerful solutions $\rightarrow$ numerical problems.
  - ill-posed problems.
  - Bayesian Prior.
  - SVM: Least Complex without classification error.

Automatica 2011, CDC 2010, ...

Batch Setting - MINLIP

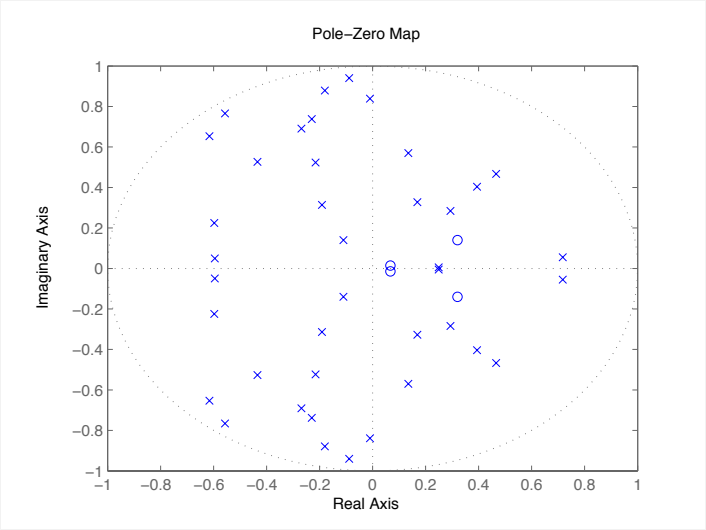


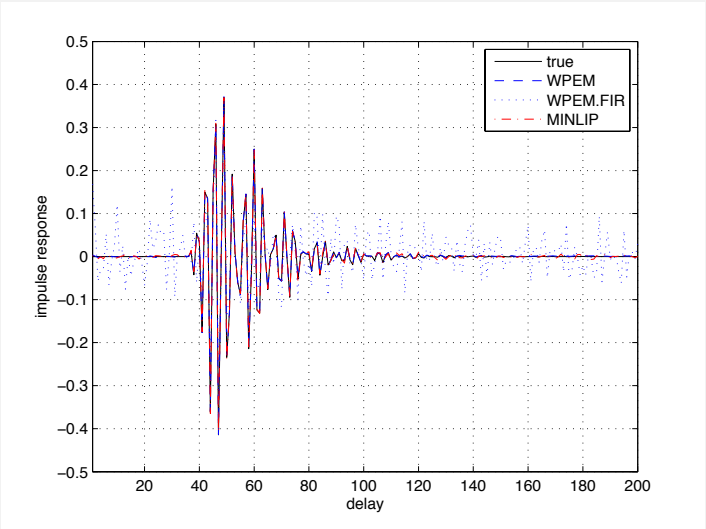


$$\arg\min_{\mathbf{h}} \|\mathbf{h}\| \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

- Example.
- Almost Consistency (Lipschitz).
- Extensions to Noise.
- Quantization (Margin).

Batch Setting - MINLIP

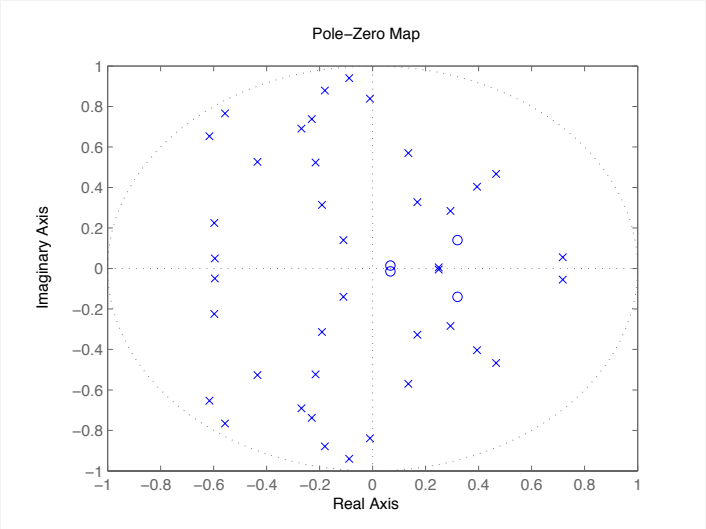


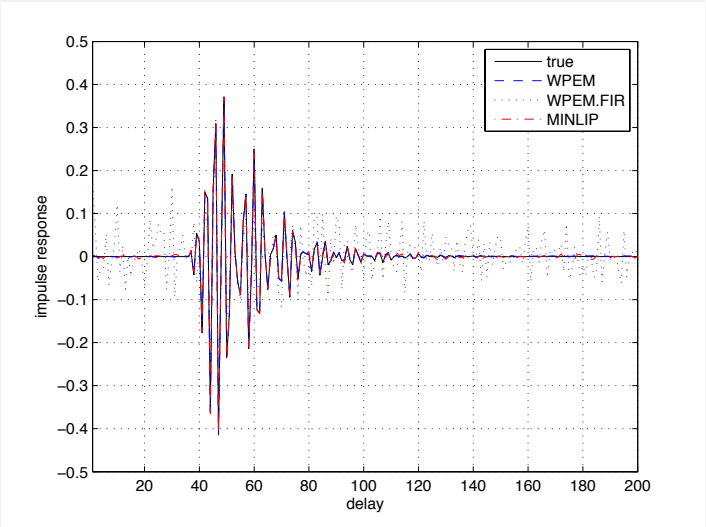


$$\arg\min_{\mathbf{h}} \|\mathbf{h}\| \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

- Example.
- Almost Consistency (Lipschitz).
- Extensions to Noise.
- Quantization (Margin).

Batch Setting - MINLIP

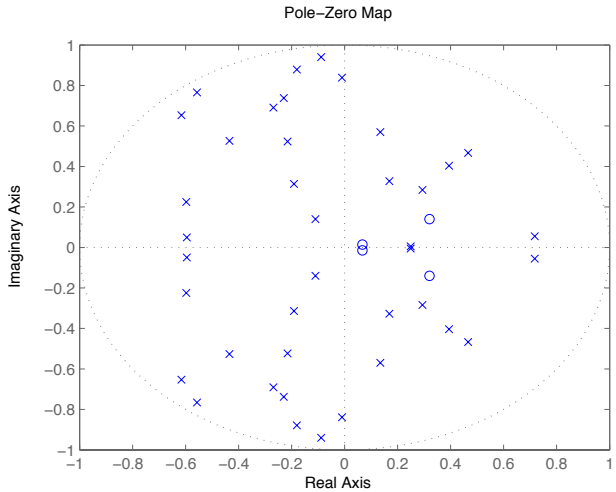


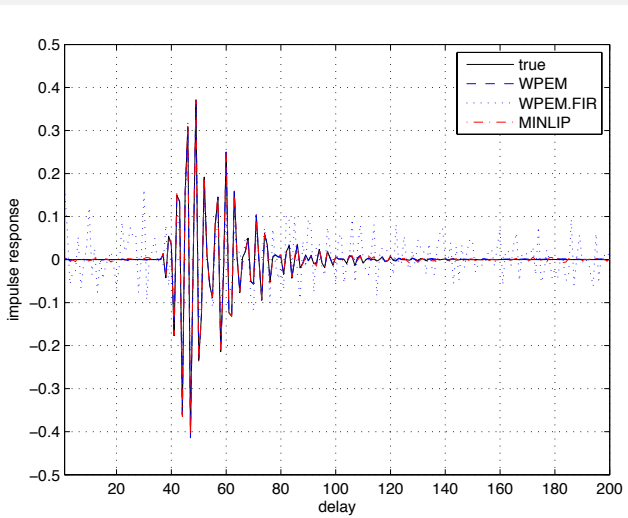


$$\arg\min_{\mathbf{h}} \|\mathbf{h}\| \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

- Example.
- Almost Consistency (Lipschitz).
- Extensions to Noise.
- Quantization (Margin).

Batch Setting - MINLIP

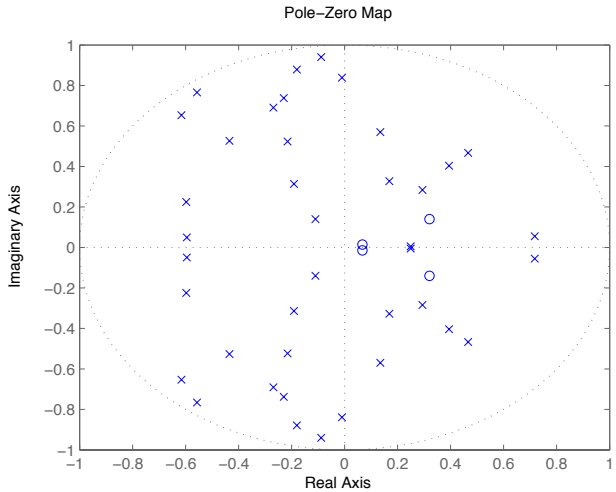


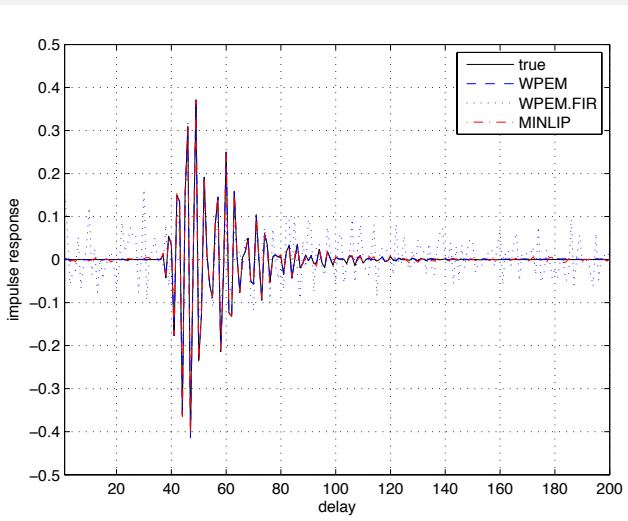


$$\arg\min_{\mathbf{h}} \|\mathbf{h}\| \text{ s.t. } (y_i - y_j) \leq \mathbf{h}^T(\mathbf{u}_i - \mathbf{u}_j),$$

- Example.
- Almost Consistency (Lipschitz).
- Extensions to Noise.
- Quantization (Margin).

Batch Setting - MINLIP



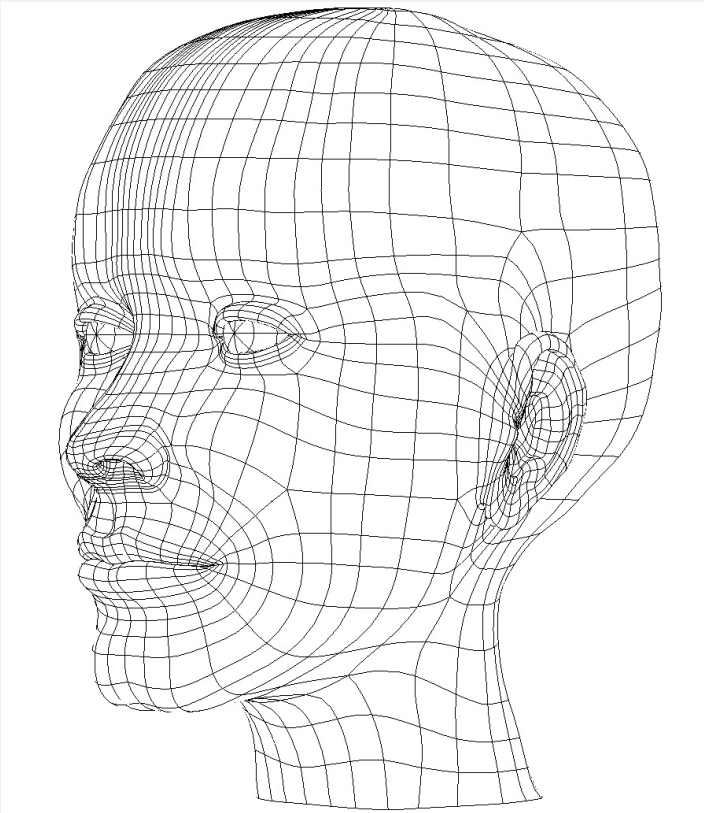


Definition (Local PE)

For every $\mathbf{u}$ one has m linear independent vectors $\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$ where

$$\|\mathbf{u} - \mathbf{u}_i\|_2 \leq \epsilon$$

Recursive Identification - RANKTRON

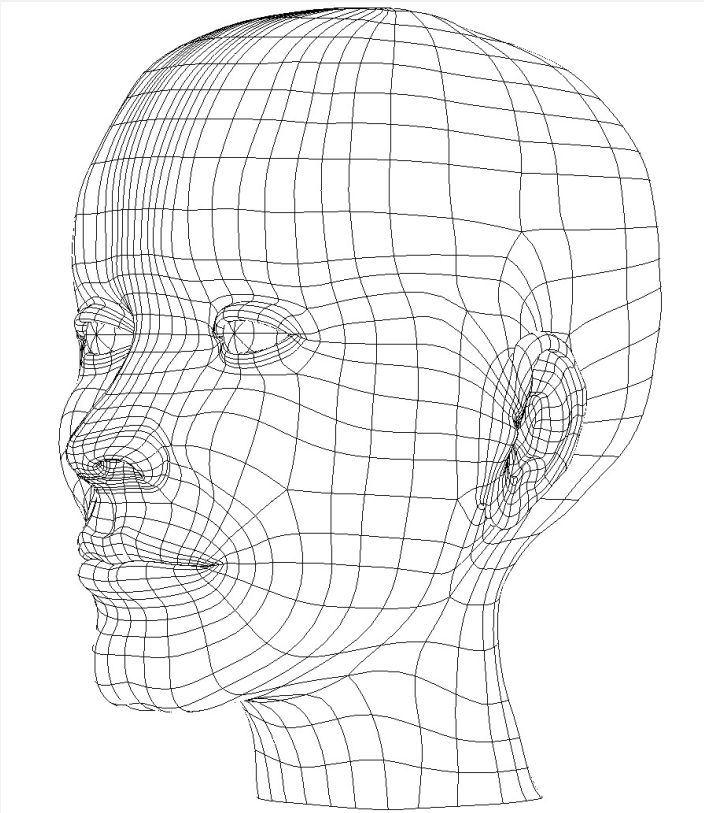


Protocol:

1. Initialize $(f_{(0)}, \mathbf{h}_{(0)})$.
2. Observe (!) new u_t.
3. Predict y_t using $\hat{y}_t = f_{(t-1)}(\mathbf{h}_{(t-1)}^T \mathbf{u}_t)$.
4. Observe y_t, and score $e_t = (y_t - \hat{y}_t)$.
5. Update $(f_{(t-1)}, \mathbf{h}_{(t-1)}) \rightarrow (f_{(t)}, \mathbf{h}_{(t)})$ using $e_t, \mathbf{u}_t$.
6. Iterate (2-4) for $t = 1, 2, \ldots$.

Cfr. PERCEPTRON (1950s)

Recursive Identification - RANKTRON

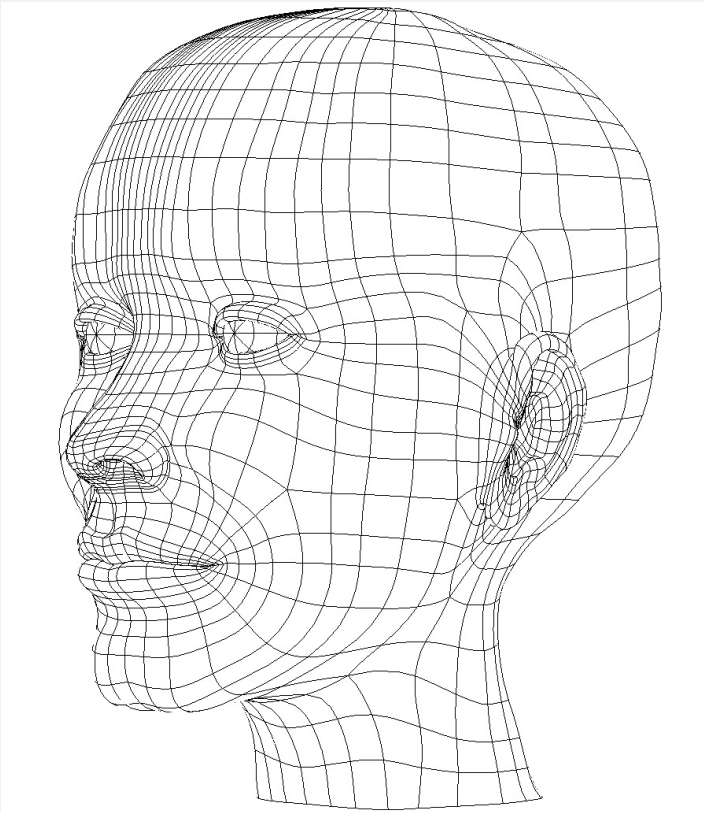


Protocol:

1. Initialize $(f_{(0)}, \mathbf{h}_{(0)})$.
2. Observe (!) new u_t.
3. Predict y_t using $\hat{y}_t = f_{(t-1)}(\mathbf{h}_{(t-1)}^T \mathbf{u}_t)$.
4. Observe y_t, and score $e_t = (y_t - \hat{y}_t)$.
5. Update $(f_{(t-1)}, \mathbf{h}_{(t-1)}) \rightarrow (f_{(t)}, \mathbf{h}_{(t)})$ using $e_t, \mathbf{u}_t$.
6. Iterate (2-4) for $t = 1, 2, \ldots$.

Cfr. PERCEPTRON (1950s)

Recursive Identification - RANKTRON

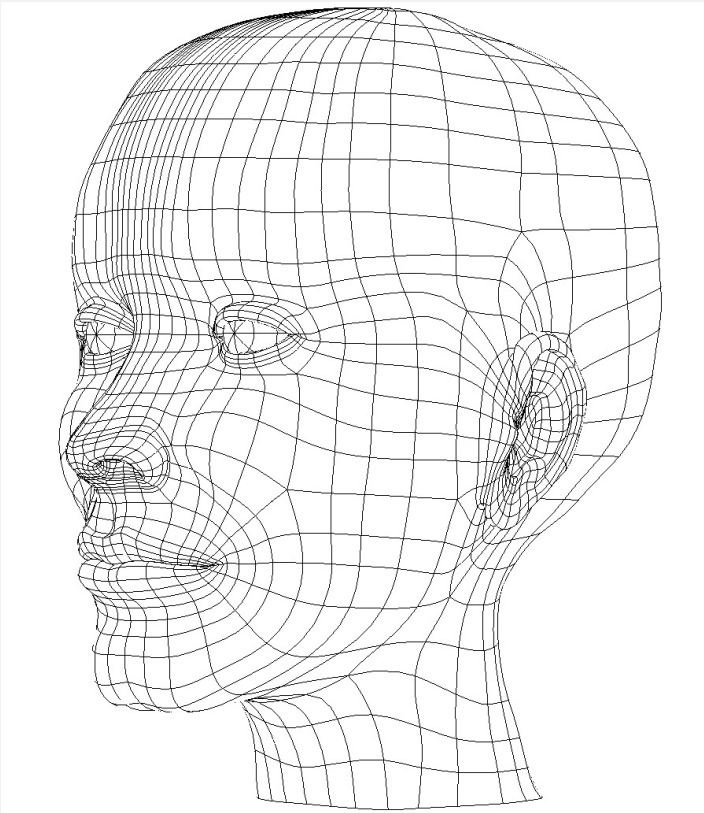


Protocol:

1. Initialize $(f_{(0)}, \mathbf{h}_{(0)})$.
2. Observe (!) new u_t.
3. Predict y_t using $\hat{y}_t = f_{(t-1)}(\mathbf{h}_{(t-1)}^T \mathbf{u}_t)$.
4. Observe y_t, and score $e_t = (y_t - \hat{y}_t)$.
5. Update $(f_{(t-1)}, \mathbf{h}_{(t-1)}) \rightarrow (f_{(t)}, \mathbf{h}_{(t)})$ using $e_t, \mathbf{u}_t$.
6. Iterate (2-4) for $t = 1, 2, \ldots$.

Cfr. PERCEPTRON (1950s)

Recursive Identification - RANKTRON

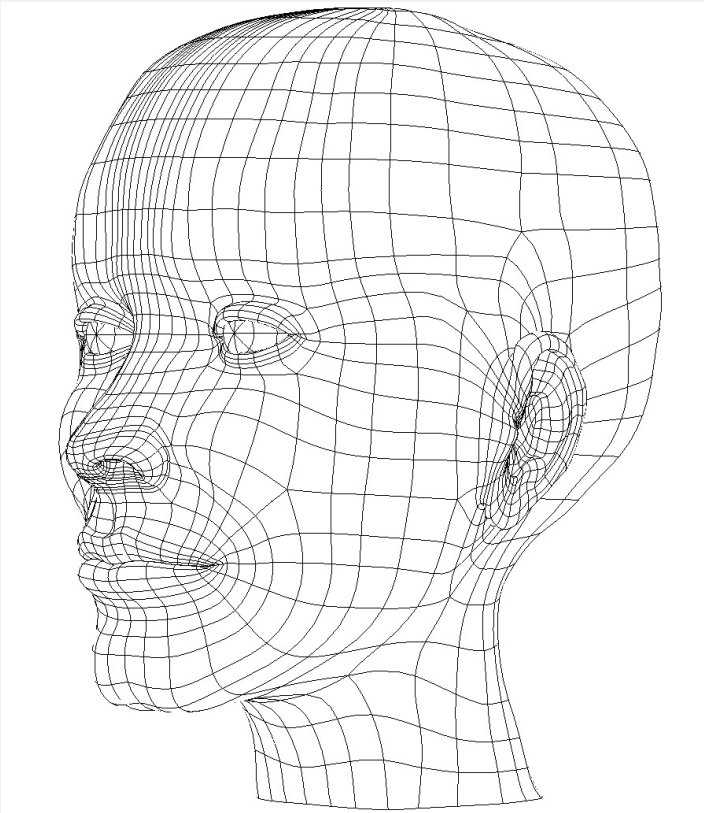


Protocol:

1. Initialize $(f_{(0)}, \mathbf{h}_{(0)})$.
2. Observe (!) new u_t.
3. Predict y_t using $\hat{y}_t = f_{(t-1)}(\mathbf{h}_{(t-1)}^T \mathbf{u}_t)$.
4. Observe y_t, and score $e_t = (y_t - \hat{y}_t)$.
5. Update $(f_{(t-1)}, \mathbf{h}_{(t-1)}) \rightarrow (f_{(t)}, \mathbf{h}_{(t)})$ using $e_t, \mathbf{u}_t$.
6. Iterate (2-4) for $t = 1, 2, \ldots$.

Cfr. PERCEPTRON (1950s)

Recursive Identification - RANKTRON

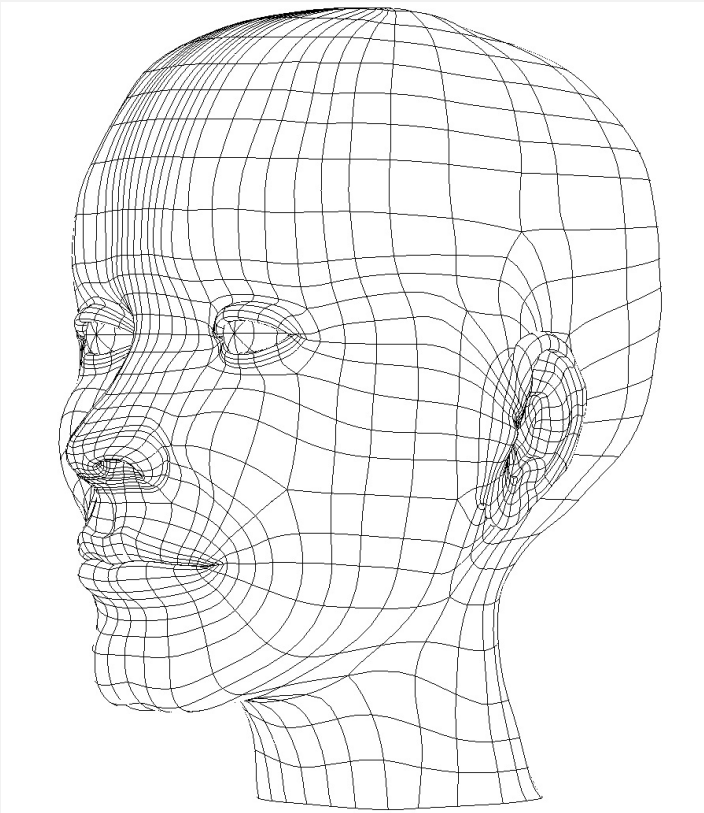


Protocol:

1. Initialize $(f_{(0)}, \mathbf{h}_{(0)})$.
2. Observe (!) new u_t.
3. Predict y_t using $\hat{y}_t = f_{(t-1)}(\mathbf{h}_{(t-1)}^T \mathbf{u}_t)$.
4. Observe y_t, and score $e_t = (y_t - \hat{y}_t)$.
5. Update $(f_{(t-1)}, \mathbf{h}_{(t-1)}) \rightarrow (f_{(t)}, \mathbf{h}_{(t)})$ using $e_t, \mathbf{u}_t$.
6. Iterate (2-4) for $t = 1, 2, \ldots$.

Cfr. PERCEPTRON (1950s)

Recursive Identification - RANKTRON

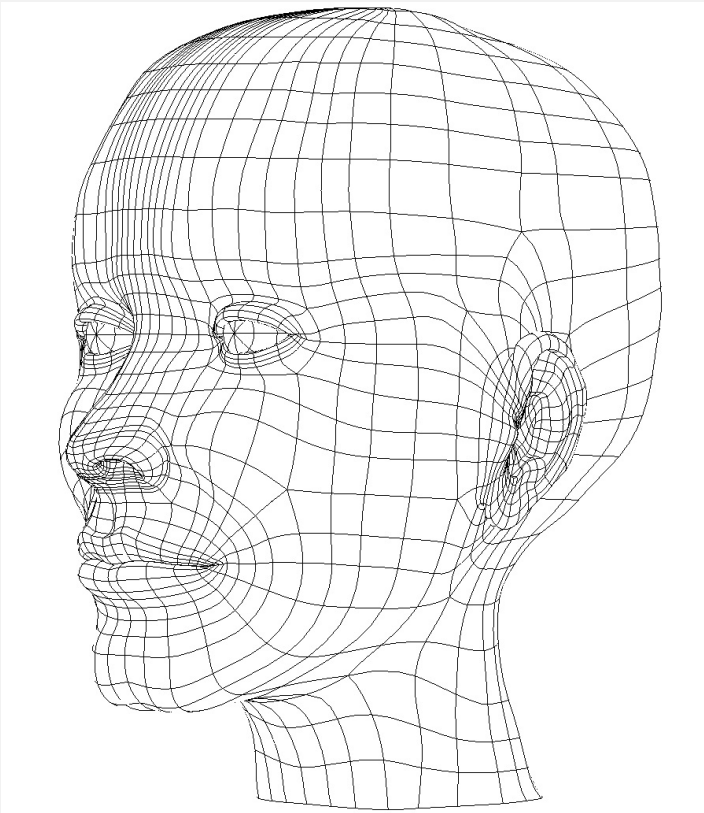


Protocol:

1. Initialize $(f_{(0)}, \mathbf{h}_{(0)})$.
2. Observe (!) new u_t.
3. Predict y_t using $\hat{y}_t = f_{(t-1)}(\mathbf{h}_{(t-1)}^T \mathbf{u}_t)$.
4. Observe y_t, and score $e_t = (y_t - \hat{y}_t)$.
5. Update $(f_{(t-1)}, \mathbf{h}_{(t-1)}) \rightarrow (f_{(t)}, \mathbf{h}_{(t)})$ using $e_t, \mathbf{u}_t$.
6. Iterate (2-4) for $t = 1, 2, \ldots$.

Cfr. PERCEPTRON (1950s)

Recursive Identification - RANKTRON

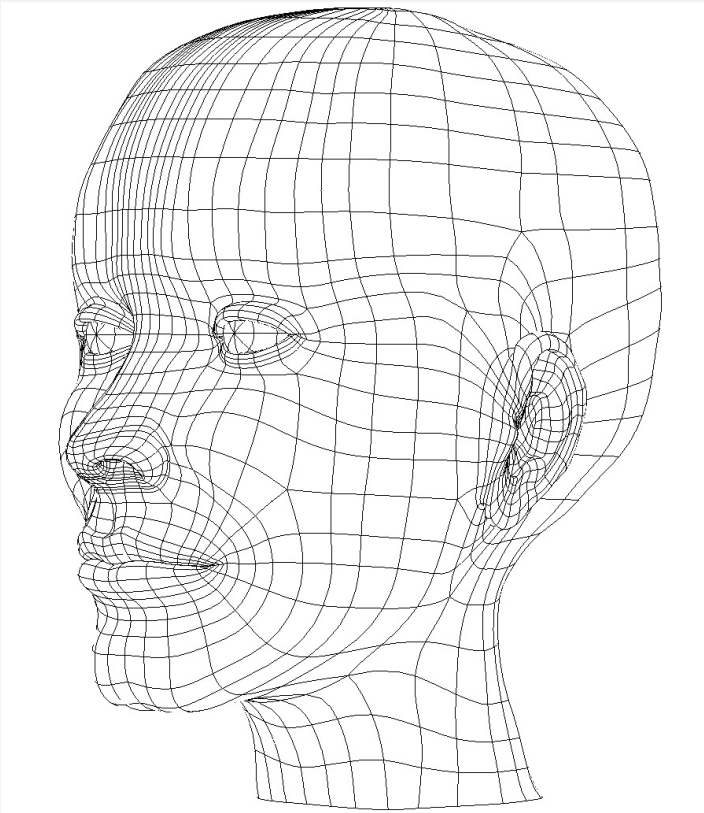


Protocol:

1. Initialize $(f_{(0)}, \mathbf{h}_{(0)})$.
2. Observe (!) new u_t.
3. Predict y_t using $\hat{y}_t = f_{(t-1)}(\mathbf{h}_{(t-1)}^T \mathbf{u}_t)$.
4. Observe y_t, and score $e_t = (y_t - \hat{y}_t)$.
5. Update $(f_{(t-1)}, \mathbf{h}_{(t-1)}) \rightarrow (f_{(t)}, \mathbf{h}_{(t)})$ using $e_t, \mathbf{u}_t$.
6. Iterate (2-4) for $t = 1, 2, \ldots$.

Cfr. PERCEPTRON (1950s)

Recursive Identification - RANKTRON

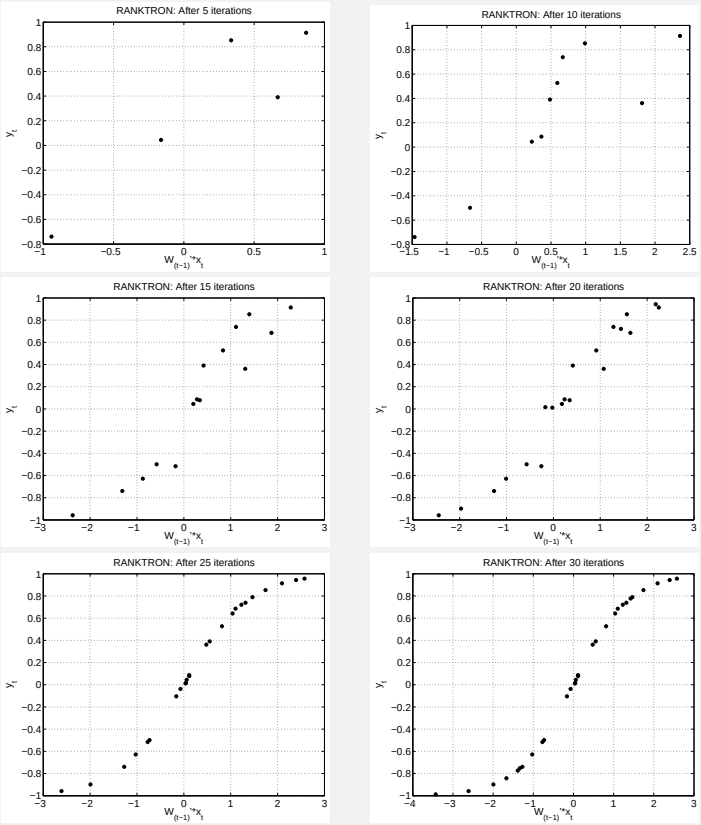


Protocol:

1. Initialize $(f_{(0)}, \mathbf{h}_{(0)})$.
2. Observe (!) new u_t.
3. Predict y_t using $\hat{y}_t = f_{(t-1)}(\mathbf{h}_{(t-1)}^T \mathbf{u}_t)$.
4. Observe y_t, and score $e_t = (y_t - \hat{y}_t)$.
5. Update $(f_{(t-1)}, \mathbf{h}_{(t-1)}) \rightarrow (f_{(t)}, \mathbf{h}_{(t)})$ using $e_t, \mathbf{u}_t$.
6. Iterate (2-4) for $t = 1, 2, \ldots$.

Cfr. PERCEPTRON (1950s)

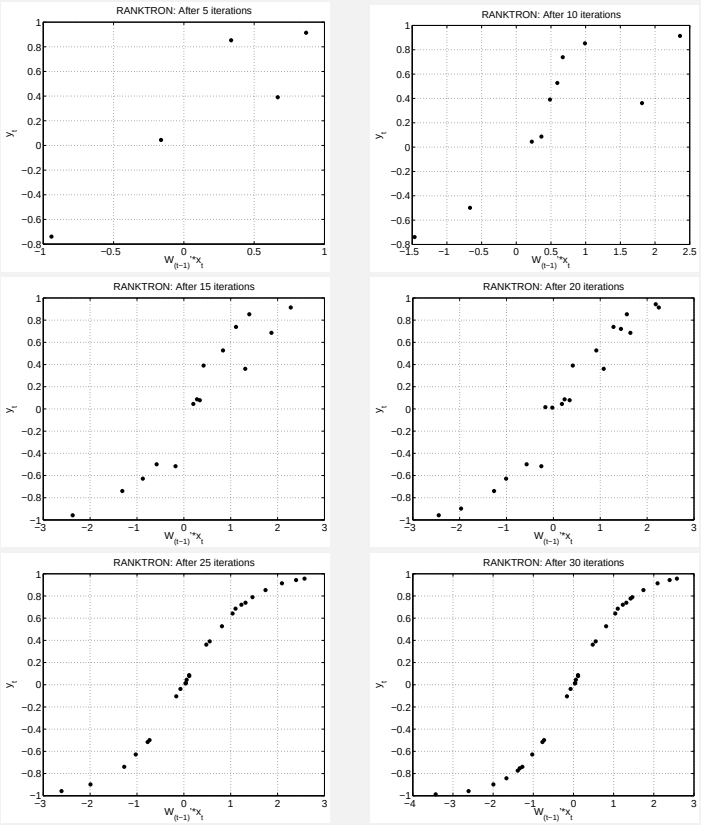
Recursive Identification - RANKTRON



- Proof that $\sum_t \ell(e_t)$ not too large.
- Speed controlled by $\|\mathbf{h}_0\|_2$.
- Proof that $(f_{(t)}, \mathbf{h}_{(t)}) \rightarrow (f_0, \mathbf{h}_0)$.
- RANKTRON.v1: Identify $\mathbf{h}_0$.
- RANKTRON.v2: Identify $(f_0, \mathbf{h}_0)$.
- Piecewise linear representation of f_0.
- ORACLE bounds.
- Tracking bounds.
- Example.

[Under Revision 2010-2011]

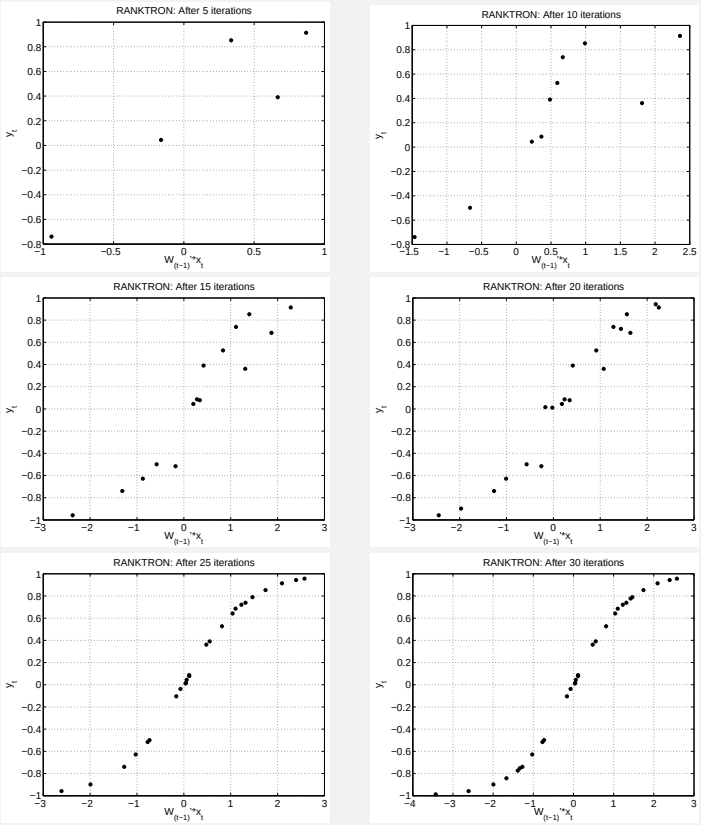
Recursive Identification - RANKTRON



- Proof that $\sum_t \ell(e_t)$ not too large.
- Speed controlled by $\|\mathbf{h}_0\|_2$.
- Proof that $(f_{(t)}, \mathbf{h}_{(t)}) \rightarrow (f_0, \mathbf{h}_0)$.
- RANKTRON.v1: Identify $\mathbf{h}_0$.
- RANKTRON.v2: Identify $(f_0, \mathbf{h}_0)$.
- Piecewise linear representation of f_0.
- ORACLE bounds.
- Tracking bounds.
- Example.

[Under Revision 2010-2011]

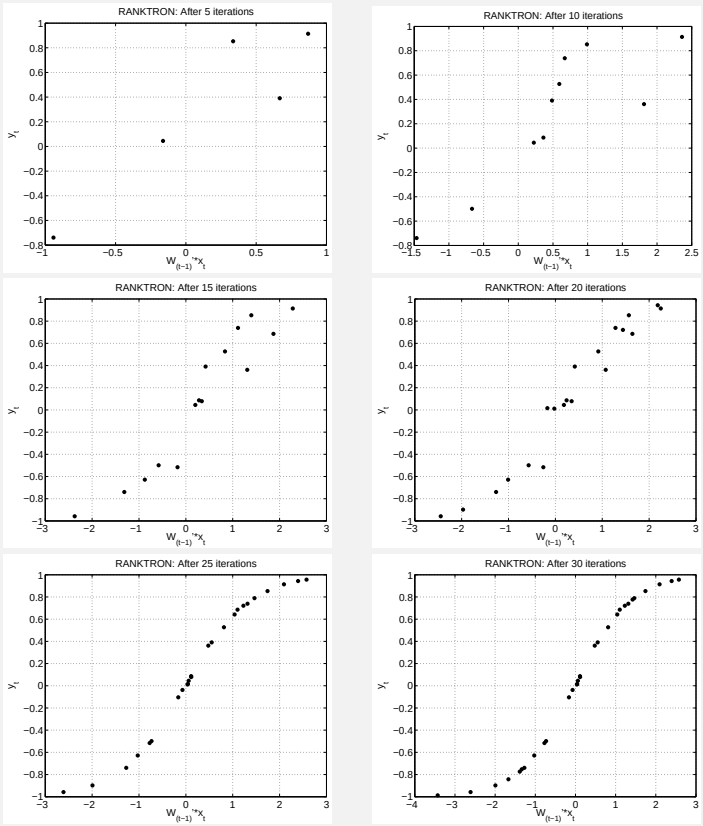
Recursive Identification - RANKTRON



- Proof that $\sum_t \ell(e_t)$ not too large.
- Speed controlled by $\|\mathbf{h}_0\|_2$.
- Proof that $(f_{(t)}, \mathbf{h}_{(t)}) \rightarrow (f_0, \mathbf{h}_0)$.
- RANKTRON.v1: Identify $\mathbf{h}_0$.
- RANKTRON.v2: Identify $(f_0, \mathbf{h}_0)$.
- Piecewise linear representation of f_0.
- ORACLE bounds.
- Tracking bounds.
- Example.

[Under Revision 2010-2011]

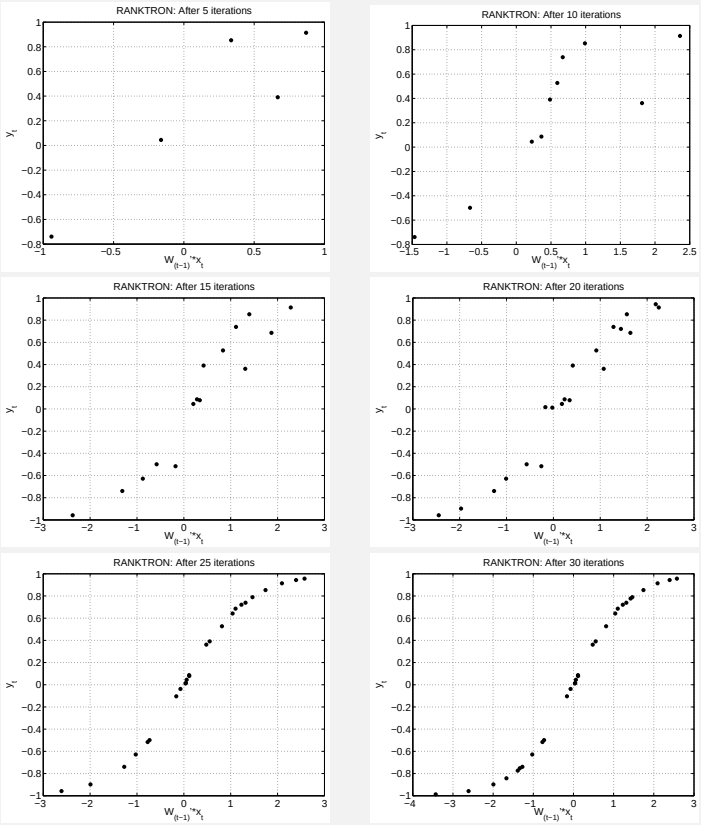
Recursive Identification - RANKTRON



- Proof that $\sum_t \ell(e_t)$ not too large.
- Speed controlled by $\|\mathbf{h}_0\|_2$.
- Proof that $(f_{(t)}, \mathbf{h}_{(t)}) \rightarrow (f_0, \mathbf{h}_0)$.
- RANKTRON.v1: Identify $\mathbf{h}_0$.
- RANKTRON.v2: Identify $(f_0, \mathbf{h}_0)$.
- Piecewise linear representation of f_0.
- ORACLE bounds.
- Tracking bounds.
- Example.

[Under Revision 2010-2011]

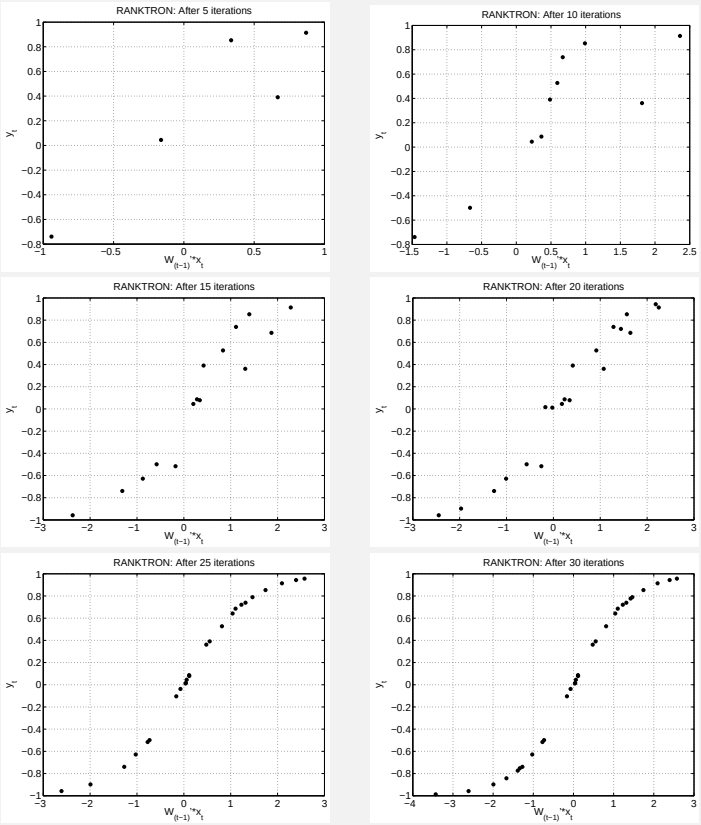
Recursive Identification - RANKTRON



- Proof that $\sum_t \ell(e_t)$ not too large.
- Speed controlled by $\|\mathbf{h}_0\|_2$.
- Proof that $(f_{(t)}, \mathbf{h}_{(t)}) \rightarrow (f_0, \mathbf{h}_0)$.
- RANKTRON.v1: Identify $\mathbf{h}_0$.
- RANKTRON.v2: Identify $(f_0, \mathbf{h}_0)$.
- Piecewise linear representation of f_0.
- ORACLE bounds.
- Tracking bounds.
- Example.

[Under Revision 2010-2011]

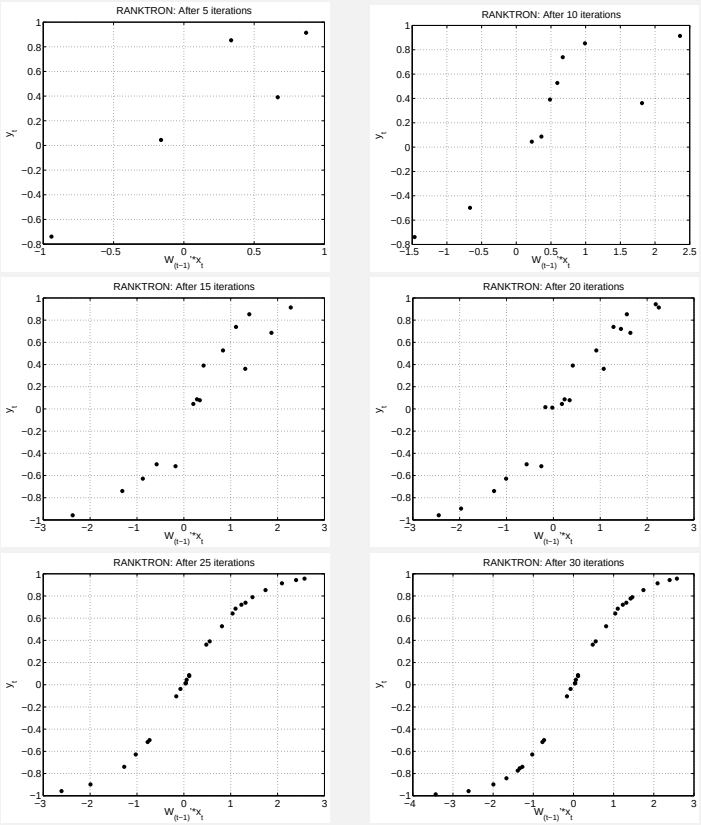
Recursive Identification - RANKTRON



- Proof that $\sum_t \ell(e_t)$ not too large.
- Speed controlled by $\|\mathbf{h}_0\|_2$.
- Proof that $(f_{(t)}, \mathbf{h}_{(t)}) \rightarrow (f_0, \mathbf{h}_0)$.
- RANKTRON.v1: Identify $\mathbf{h}_0$.
- RANKTRON.v2: Identify $(f_0, \mathbf{h}_0)$.
- Piecewise linear representation of f_0.
- ORACLE bounds.
- Tracking bounds.
- Example.

[Under Revision 2010-2011]

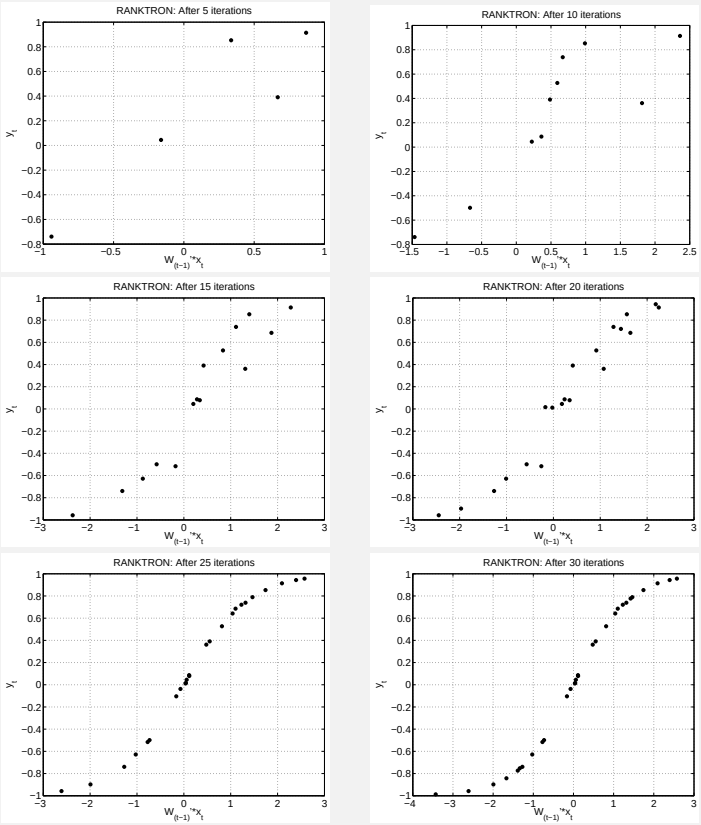
Recursive Identification - RANKTRON



- Proof that $\sum_t \ell(e_t)$ not too large.
- Speed controlled by $\|\mathbf{h}_0\|_2$.
- Proof that $(f_{(t)}, \mathbf{h}_{(t)}) \rightarrow (f_0, \mathbf{h}_0)$.
- RANKTRON.v1: Identify $\mathbf{h}_0$.
- RANKTRON.v2: Identify $(f_0, \mathbf{h}_0)$.
- Piecewise linear representation of f_0.
- ORACLE bounds.
- Tracking bounds.
- Example.

[Under Revision 2010-2011]

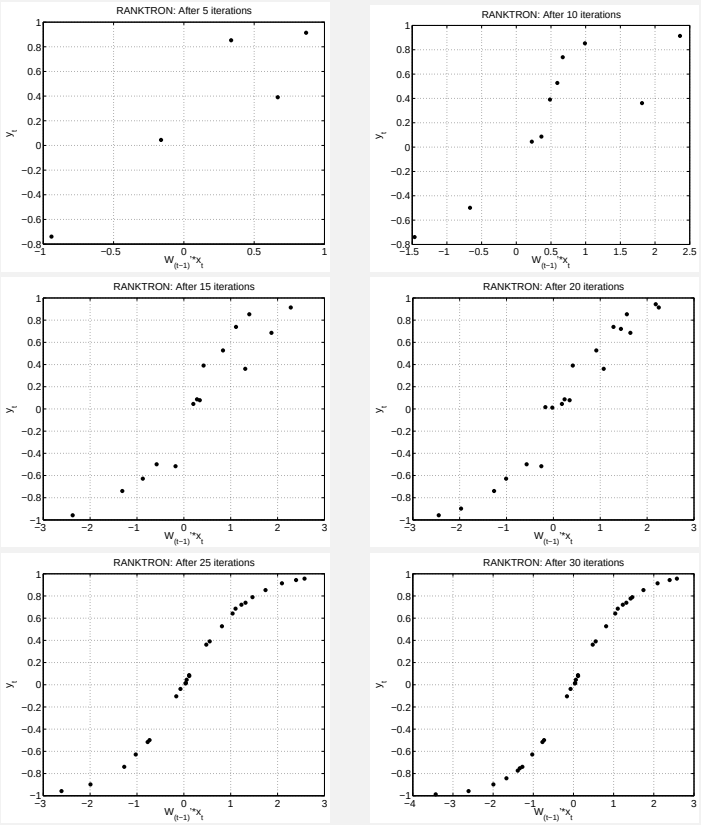
Recursive Identification - RANKTRON



- Proof that $\sum_t \ell(e_t)$ not too large.
- Speed controlled by $\|\mathbf{h}_0\|_2$.
- Proof that $(f_{(t)}, \mathbf{h}_{(t)}) \rightarrow (f_0, \mathbf{h}_0)$.
- RANKTRON.v1: Identify $\mathbf{h}_0$.
- RANKTRON.v2: Identify $(f_0, \mathbf{h}_0)$.
- Piecewise linear representation of f_0.
- ORACLE bounds.
- Tracking bounds.
- Example.

[Under Revision 2010-2011]

Recursive Identification - RANKTRON



- Proof that $\sum_t \ell(e_t)$ not too large.
- Speed controlled by $\|\mathbf{h}_0\|_2$.
- Proof that $(f_{(t)}, \mathbf{h}_{(t)}) \rightarrow (f_0, \mathbf{h}_0)$.
- RANKTRON.v1: Identify $\mathbf{h}_0$.
- RANKTRON.v2: Identify $(f_0, \mathbf{h}_0)$.
- Piecewise linear representation of f_0.
- ORACLE bounds.
- Tracking bounds.
- Example.

[Under Revision 2010-2011]

Non-parameteric Control - NORTKNAR.

Q : Steer plant with $G(q^-1) = f_0(\mathbf{h}_0^T \mathbf{u}_t)$ to $\alpha \in \mathbb{R}$ by manipulating input $\{u_1, \ldots, u_t, \ldots\}$.

D : Totally Model-free: no need for first identifying $(f_0, \mathbf{h}_0)$ in *any form*.

- Initialize $t = 0$.
- Compute u_t.
- Apply u_t to the actual Plant.
- Compare expected outcome $\alpha \in \mathbb{R}$ with observation y_t.
- Iterate (1-3) for $t = 1, 2, ...$

Non-parameteric Control - NORTKNAR.

Q : Steer plant with $G(q^-1) = f_0(\mathbf{h}_0^T \mathbf{u}_t)$ to $\alpha \in \mathbb{R}$ by manipulating input $\{u_1, \ldots, u_t, \ldots\}$.

D : Totally Model-free: no need for first identifying $(f_0, \mathbf{h}_0)$ in *any form*.

- Initialize $t = 0$.
- Compute u_t.
- Apply u_t to the actual Plant.
- Compare expected outcome $\alpha \in \mathbb{R}$ with observation y_t.
- Iterate (1-3) for $t = 1, 2, ...$

Non-parameteric Control - NORTKNAR.

Q : Steer plant with $G(q^-1) = f_0(\mathbf{h}_0^T \mathbf{u}_t)$ to $\alpha \in \mathbb{R}$ by manipulating input $\{u_1, \ldots, u_t, \ldots\}$.

D : Totally Model-free: no need for first identifying $(f_0, \mathbf{h}_0)$ in *any form*.

- Initialize $t = 0$.
- Compute u_t.
- Apply u_t to the actual Plant.
- Compare expected outcome $\alpha \in \mathbb{R}$ with observation y_t.
- Iterate (1-3) for $t = 1, 2, ...$

Non-parameteric Control - NORTKNAR.

Q : Steer plant with $G(q^-1) = f_0(\mathbf{h}_0^T \mathbf{u}_t)$ to $\alpha \in \mathbb{R}$ by manipulating input $\{u_1, \ldots, u_t, \ldots\}$.

D : Totally Model-free: no need for first identifying $(f_0, \mathbf{h}_0)$ in *any form*.

- Initialize $t = 0$.
- Compute u_t.
- Apply u_t to the actual Plant.
- Compare expected outcome $\alpha \in \mathbb{R}$ with observation y_t.
- Iterate (1-3) for $t = 1, 2, \ldots$.

Non-parameteric Control - NORTKNAR.

Q : Steer plant with $G(q^-1) = f_0(\mathbf{h}_0^T \mathbf{u}_t)$ to $\alpha \in \mathbb{R}$ by manipulating input $\{u_1, \ldots, u_t, \ldots\}$.

D : Totally Model-free: no need for first identifying $(f_0, \mathbf{h}_0)$ in *any form*.

- Initialize $t = 0$.
- Compute u_t.
- Apply u_t to the actual Plant.
- Compare expected outcome $\alpha \in \mathbb{R}$ with observation y_t.
- Iterate (1-3) for $t = 1, 2, \ldots$.

Non-parameteric Control - NORTKNAR.

Q : Steer plant with $G(q^-1) = f_0(\mathbf{h}_0^T \mathbf{u}_t)$ to $\alpha \in \mathbb{R}$ by manipulating input $\{u_1, \ldots, u_t, \ldots\}$.

D : Totally Model-free: no need for first identifying $(f_0, \mathbf{h}_0)$ in *any form*.

- Initialize $t = 0$.
- Compute u_t.
- Apply u_t to the actual Plant.
- Compare expected outcome $\alpha \in \mathbb{R}$ with observation y_t.
- Iterate (1-3) for $t = 1, 2, \ldots$.

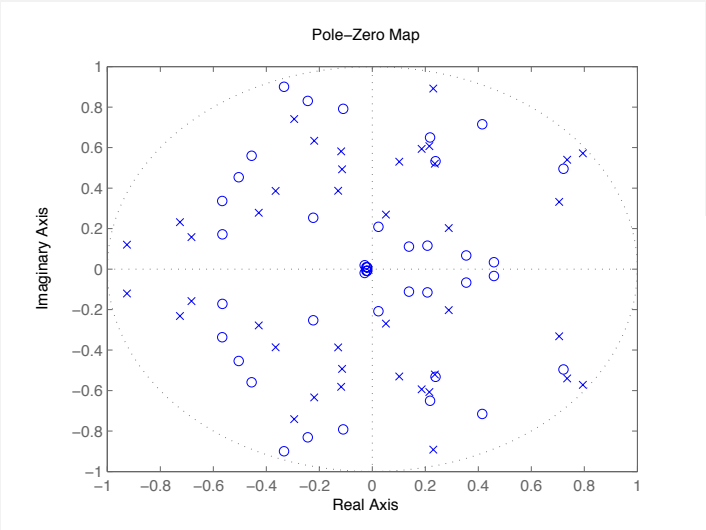
Non-parameteric Control - NORTKNAR.

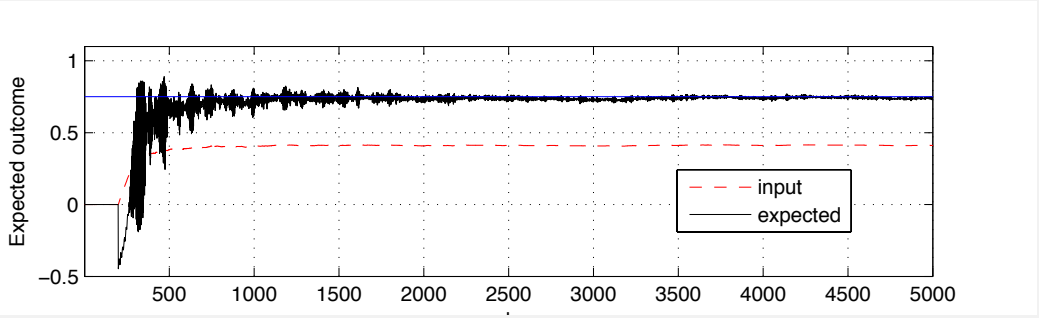
Q : Steer plant with $G(q^-1) = f_0(\mathbf{h}_0^T \mathbf{u}_t)$ to $\alpha \in \mathbb{R}$ by manipulating input $\{u_1, \ldots, u_t, \ldots\}$.

D : Totally Model-free: no need for first identifying $(f_0, \mathbf{h}_0)$ in *any form*.

- Initialize $t = 0$.
- Compute u_t.
- Apply u_t to the actual Plant.
- Compare expected outcome $\alpha \in \mathbb{R}$ with observation y_t.
- Iterate (1-3) for $t = 1, 2, ...$

Non-parameteric Control - NORTKNAR.

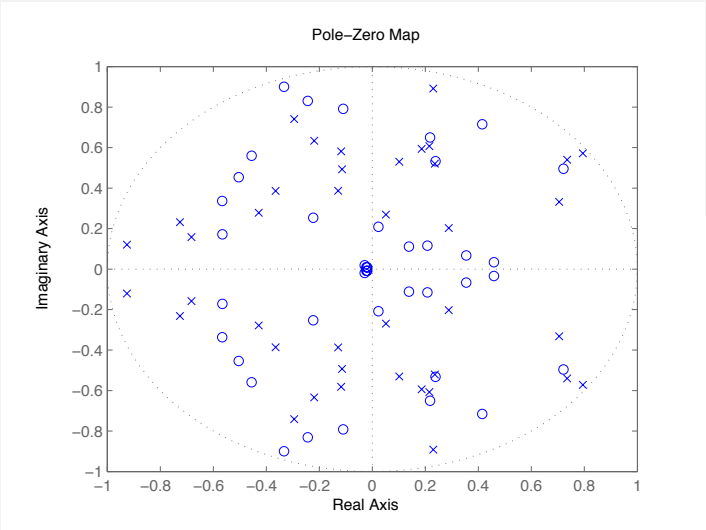


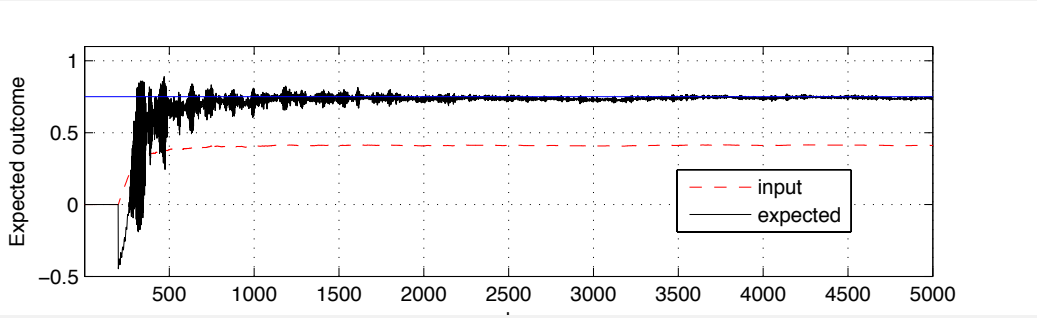


$$u_t = u_{t-1} + \eta_t(\alpha - y_t)$$

- Robbins-Monro, but.
- Converge if:
  - objective achievable (α in range).
  - $1_d^T \mathbf{h}_0 > 0$
  - $\sum_t \eta_t \to \infty, \sum_t \eta_t^2 \to O(1)$
- Example.

Non-parameteric Control - NORTKNAR.

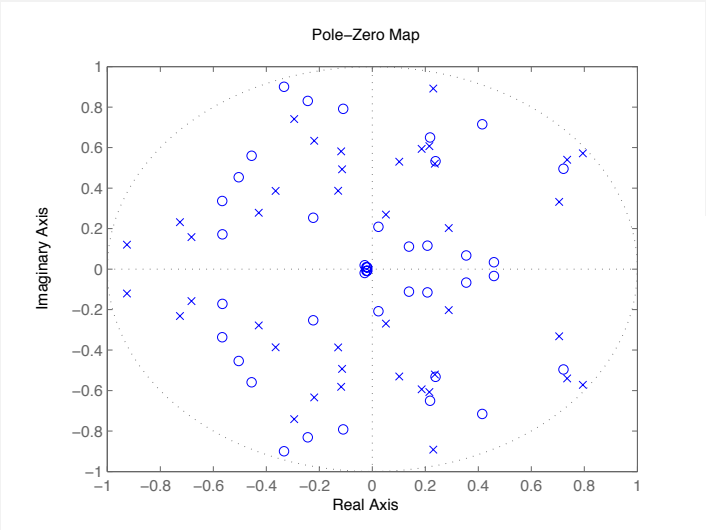


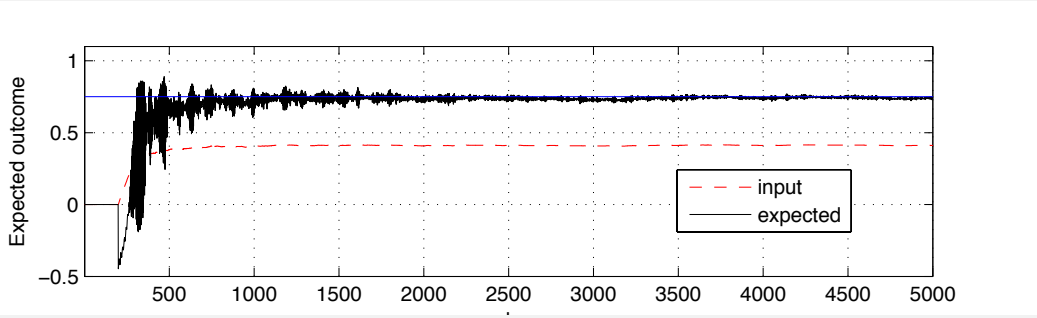


$$u_t = u_{t-1} + \eta_t(\alpha - y_t)$$

- Robbins-Monro, but.
- Converge if:
  - objective achievable (α in range).
  - $1_d^T \mathbf{h}_0 > 0$
  - $\sum_t \eta_t \to \infty, \sum_t \eta_t^2 \to O(1)$
- Example.

Non-parameteric Control - NORTKNAR.

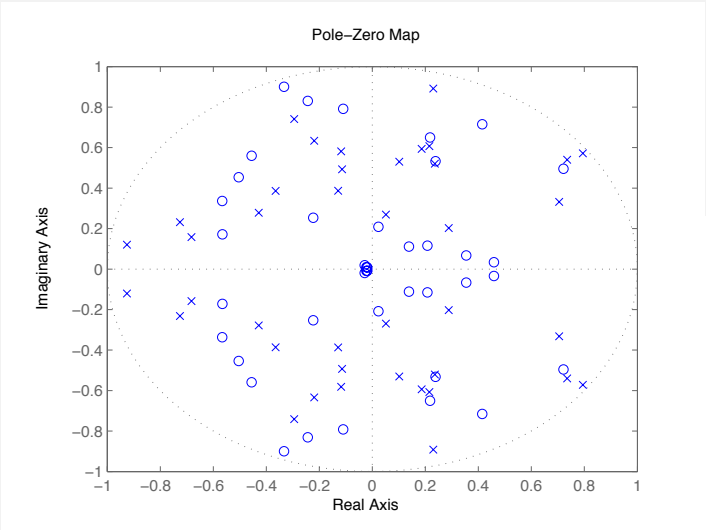


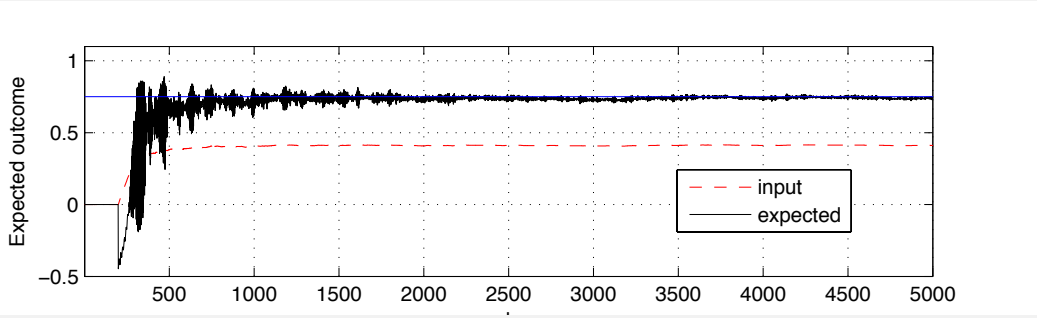


$$u_t = u_{t-1} + \eta_t(\alpha - y_t)$$

- Robbins-Monro, but.
- Converge if:
  - objective achievable (α in range).
  - $1_d^T \mathbf{h}_0 > 0$
  - $\sum_t \eta_t \to \infty, \sum_t \eta_t^2 \to O(1)$
- Example.

Non-parameteric Control - NORTKNAR.

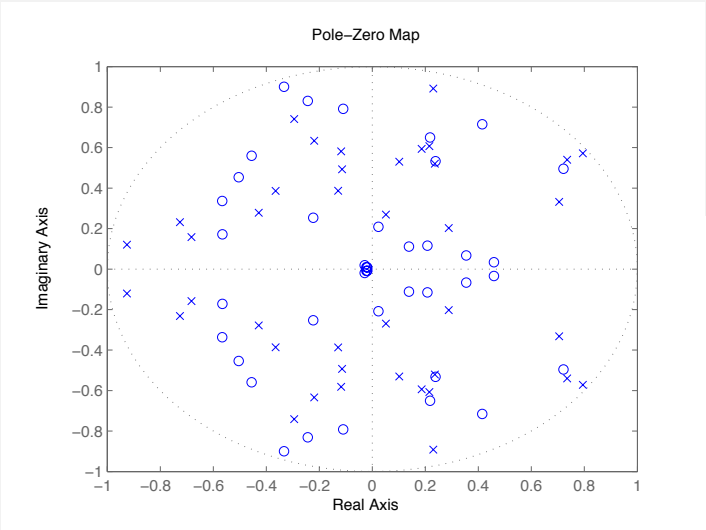


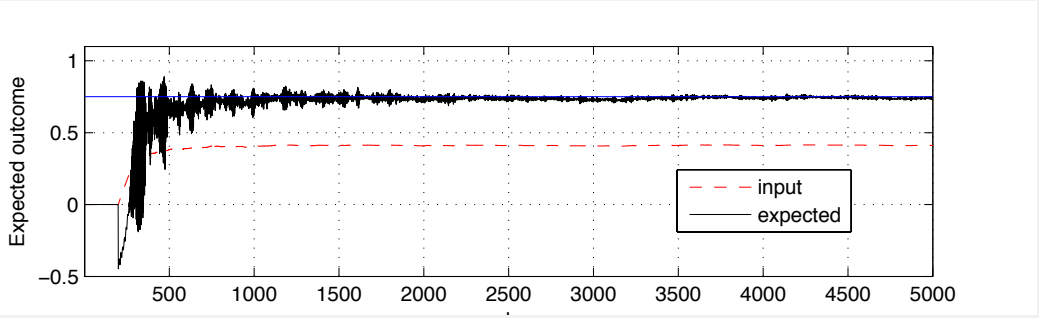


$$u_t = u_{t-1} + \eta_t(\alpha - y_t)$$

- Robbins-Monro, but.
- Converge if:
  - objective achievable (α in range).
  - $1_d^T \mathbf{h}_0 > 0$
  - $\sum_t \eta_t \to \infty, \sum_t \eta_t^2 \to O(1)$
- Example.

Non-parameteric Control - NORTKNAR.

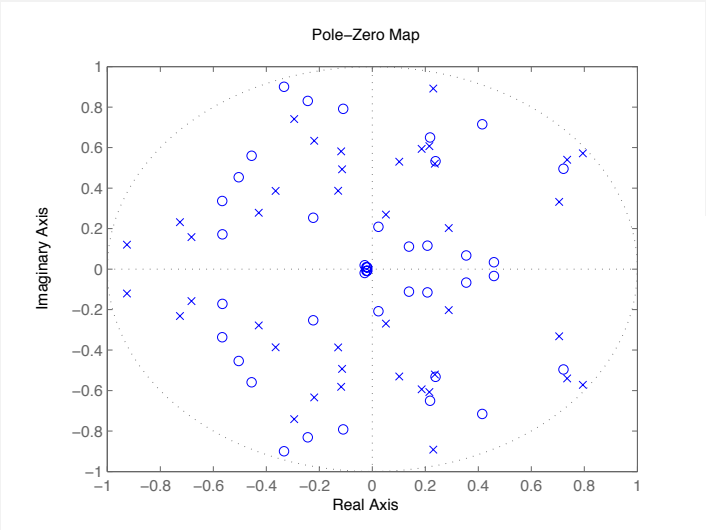


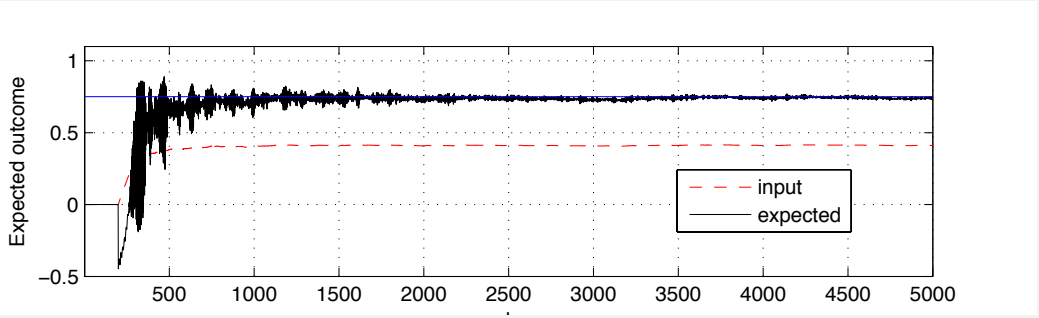


$$u_t = u_{t-1} + \eta_t(\alpha - y_t)$$

- Robbins-Monro, but.
- Converge if:
  - objective achievable (α in range).
  - $1_d^T \mathbf{h}_0 > 0$
  - $\sum_t \eta_t \to \infty, \sum_t \eta_t^2 \to O(1)$
- Example.

Non-parameteric Control - NORTKNAR.

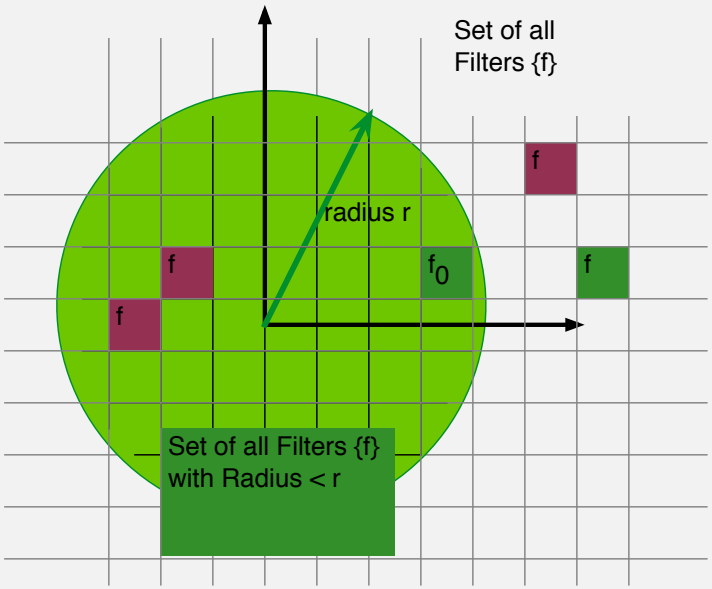




$$u_t = u_{t-1} + \eta_t(\alpha - y_t)$$

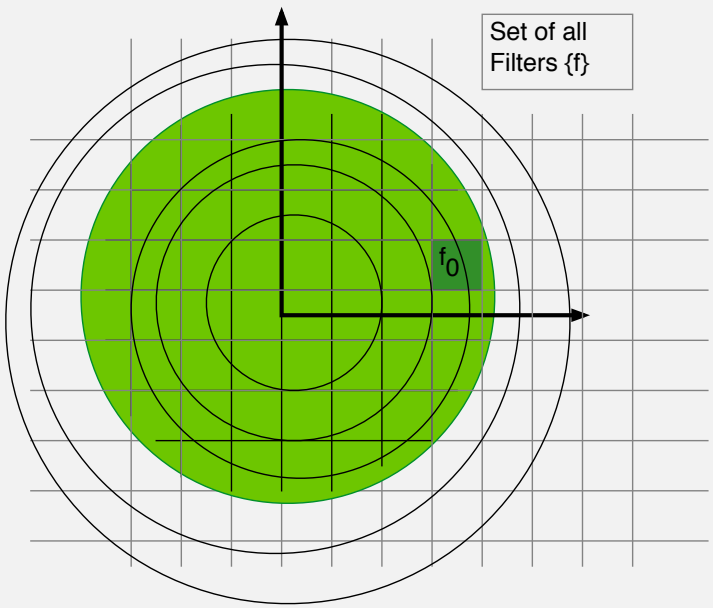
- Robbins-Monro, but.
- Converge if:
  - objective achievable (α in range).
  - $1_d^T \mathbf{h}_0 > 0$
  - $\sum_t \eta_t \to \infty, \sum_t \eta_t^2 \to O(1)$
- Example.

Complexity Control - Occam's razor revisited.



- Number of parameters?
- Number of free states?
- Desiderata:
  - Proper norm.
  - True $\mathbf{h}_0$ is low complexity.
  - Small number of f's with small complexity (covering).
  - $\mathbf{h} \sim \mathbf{h}'$, then order $\|\mathbf{h}\| < \|\mathbf{h}'\|$.
  - Only 1 $\mathbf{h}$ s.t. $\|\mathbf{h}\| = 0$.
  - f with small norm easily falsifiable.
  - Interpretation (sparseness, low rank, ...)
  - Convex.
- Examples.
  - $\|\mathbf{h}_0\|_2$
  - $\|\mathbf{h}_0\|_1$
  - $\|H(\mathbf{h}_0)\|_*$

Complexity Control - Occam's razor revisited.



"It can scarcely be denied that the supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience. AE33"

- Structure all fs in structured balls $\mathcal{H}_r$ as

$$\mathcal{H}_r = \{f : \ \|f\| \leq r\}$$

- Realizable case: true $f_0 \in \mathcal{H}_{\bar{r}}$.
- Rates $\frac{\log \bar{r}}{n}$.
- Probability error $\sim$ probability of not falsifying all fs less complex than f_0.
- Numerical trick $\rightarrow$ Principled approach.
- Limit distributions - Exponential?

Conclusions & Outlooks

- + Identification of Monotone Wiener Systems.
- + Slight extension of LTIs: exploring new ideas in a firm framework.
- + Non-parametric and Model-free Control.
- ? Noise.
- ? PE vs performance bounds.
- ? Derivation variance NORTKNAR.