**Nonlinear System Identification**

**Implementation of Fast Orthogonal Search**

**Mostafa Elhoushi**

#### Presented to: Prof. Mike Korenberg



**Electrical & Computer Engineering Department**

Winter 2012

Contents

[Introduction 3](#_Toc322957109)

[Methodology 3](#_Toc322957110)

[Finding the Best Model for Noise-Free Output 3](#_Toc322957111)

[Finding the Best Model for Noisy Output 3](#_Toc322957112)

[Various Input Generations 4](#_Toc322957113)

[Analyze Various Orders 4](#_Toc322957114)

[Results 5](#_Toc322957115)

[Model Name 5](#_Toc322957116)

[Noise Inserted 5](#_Toc322957117)

[Different Input Generations 5](#_Toc322957118)

[Varying Orders 5](#_Toc322957119)

[Linear Difference Equation 6](#_Toc322957120)

[Noise Inserted 6](#_Toc322957121)

[Different Input Generations 8](#_Toc322957122)

[Varying Orders 8](#_Toc322957123)

[Analysis 8](#_Toc322957124)

[Non-Linear Difference Equation 10](#_Toc322957125)

[Noise Inserted 11](#_Toc322957126)

[Different Input Generations 12](#_Toc322957127)

[Varying Orders 12](#_Toc322957128)

[Analysis 13](#_Toc322957129)

[LNL Cascade Model 14](#_Toc322957130)

[Noise Inserted 15](#_Toc322957131)

[Different Input Generations 16](#_Toc322957132)

[Varying Orders 17](#_Toc322957133)

[Analysis 17](#_Toc322957134)

[Non-Polynomial Equation 18](#_Toc322957135)

[Noise Inserted 19](#_Toc322957136)

[Different Input Generations 20](#_Toc322957137)

[Varying Orders 20](#_Toc322957138)

[Analysis 21](#_Toc322957139)

[Static Gyroscope Data 22](#_Toc322957140)

[Reduce De-Noising 23](#_Toc322957141)

[Different Input Generations 26](#_Toc322957142)

[Varying Orders 26](#_Toc322957143)

[Analysis 26](#_Toc322957144)

[Static Accelerometer Data 27](#_Toc322957145)

[Noise Inserted 28](#_Toc322957146)

[Different Input Generations 30](#_Toc322957147)

[Varying Orders 30](#_Toc322957148)

[Analysis 30](#_Toc322957149)

[Conclusion 31](#_Toc322957150)

[Code 32](#_Toc322957151)

[Main Code 32](#_Toc322957152)

[Input Generation 34](#_Toc322957153)

[System Models 35](#_Toc322957154)

[FOS Algorithm 37](#_Toc322957155)

[Helper Functions 40](#_Toc322957156)

# Introduction

This is a report showing my implementation of Fast Orthogonal Search (FOS) and discusses its results when applied to various systems:

* a model represented by a simple linear difference equation,
* a model represented by a more complex non-linear difference equation,
* a Linear/Non-linear/Linear (LNL) cascade model,
* a model represented by a non-polynomial equation,
* bias drift of a stationary gyroscope, and
* bias drift of s stationary accelerometer.

# Methodology

Non-linear System

FOS Model

+

+

## Finding the Best Model for Noise-Free Output

1. Generate , for where is sufficiently variable (we shall use MATLAB's function rand for uniform random distribution).
2. Use the assumed non-linear model to generate resulting output, , for .
3. **Training Phase**: Use FOS to identify several possible non-linear difference equations over the record from , where :

Assume various values for and .

1. **Selection Phase**: Compare the various models’ ability to predict the true output, , over and choose the best, by obtaining %MSE=.
2. **Evaluation Phase**: Compute the %MSE of the chosen model over .

## Finding the Best Model for Noisy Output

1. Generate zero-mean white noise, , independent of with variance equivalent to: . Try different values of .
2. **Training Phase:** Use and to find various models over .
3. **Selection Phase**: Compare the various models over by ability to predict and compare %MSE with respect to noisy output=.
4. **Evaluation Phase**: Compute the noisy-free %MSE= of the chosen model over .

## Various Input Generations

Try different methods to generate input and re-compute the corresponding . Using the chosen values of and in the Selection Phase, apply the FOS algorithm to the 1st 1000 samples of data. Analyze the effect of type of input generation on %MSE.

## Analyze Various Orders

Using the chosen values of and in the Selection Phase, apply the FOS algorithm to the 1st 1000 samples of data but with maximum order of cross-products ranging from 2 to 5. Analyze the effect of maximum cross-products on %MSE.

# Results

This section explains the methodology of analyzing each model and a description of its results.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name Code which generates the model output.  **Total Elapsed Time** = Total computation time for all 3 phases (Training Phase + Selection Phase + Evaluation Phase)  **Chosen values of K & L**: The values of K & L for the selected model in the Selection Phase.   |  |  | | --- | --- | | Figure showing:  : the true output of the model  : the output of the FOS model | Figure showing vs. |   **%MSE** = Percentage Mean Square Error of the chosen model, which is equivalent to:  Table to show the computation time needed to implement the FOS algorithm for various values of K & L.   |  |  |  | | --- | --- | --- | | K | L | Computation Time | |  |  |  |  Noise Inserted This table analyzes the effect of adding noise to the output and re-runs the whole algorithm over the 3 phases (Training Phase + Selection Phase + Evaluation Phase).  We shall try different values of P: 150, 100, 50, 25.   |  |  | | --- | --- | | P =  %MSE = | Total Elapsed Time = Total computation time for all 3 phases  %MSE w.r.t. noisy output =  Ideal %MSE = | | Figure showing:  : the true output of the model  : the output of the FOS model | Figure showing vs. |  Different Input Generations Here we set K & L to the chosen values of the Selection Phase and apply FOS to the 1st 1000 samples only and evaluate the computation time and %MSE for various generation types for input, .  The different generations we use:   * Uniform Distribution: Using MATLAB’s rand function and trying different standard deviations. * Normal Distribution: Using MATLAB’s randn function and trying different standard deviations. * Sinusoidal: Using MATLAB’s sin function and trying different frequencies and amplitudes. * Triangular: Using MATLAB’s sawtooth function and trying different widths.  Varying Orders Here we set K & L to the chosen values of the Selection Phase and apply FOS to the 1st 1000 samples only but using various orders of cross products. |

## Linear Difference Equation

% Simple Difference Equation 1

y = 1 + 0.6\*x + 0.3\*delay(x,1) + 0.4\*delay(x,2) + 0.7\*delay(x,3);

**Total Elapsed Time** = 58.313930 seconds

**Chosen values of K & L**: ,

 

**%MSE** = 2.5117×10-25 %

|  |  |  |
| --- | --- | --- |
| K | L | Computation Time |
| 10 | 12 | 4.344746 seconds |
| 3 | 15 | 4.629001 seconds |
| 12 | 8 | 5.378242 seconds |
| 10 | 12 | 6.895127 seconds |
| 10 | 19 | 10.559044 seconds |
| 8 | 19 | 8.900025 seconds |
| 6 | 5 | 1.871609 seconds |
| 14 | 6 | 5.230363 seconds |
| 8 | 3 | 1.889446 seconds |
| 7 | 10 | 4.031555 seconds |

### Noise Inserted

|  |  |
| --- | --- |
| P = 150  %MSE = 3.078% | Total Elapsed Time = 22.151039 seconds  %MSE w.r.t. noisy output = 62.19%  Ideal %MSE = 61.52% |
|  |  |
| P = 100  %MSE = 2.0079% | Total Elapsed Time = 23.151971 seconds  %MSE w.r.t. noisy output = 51.69%  Ideal %MSE = 51.30% |
|  |  |
| P = 50  %MSE = 0.45% | Total Elapsed Time = 26.159095 seconds  %MSE w.r.t. noisy output = 34.41%  Ideal %MSE = 34.15% |
|  |  |
| P = 25  %MSE = 0.2255% | Total Elapsed Time = 24.730101 seconds  %MSE w.r.t. noisy output = 20.77%  Ideal %MSE = 20.41% |
|  |  |

### Different Input Generations

Setting ,

|  |  |  |  |
| --- | --- | --- | --- |
| Input Type | Parameter | Computation Time | %MSE |
| Uniform Distribution | Standard Deviation = 1 | 1.531016 seconds | 2.67×10-25% |
| Uniform Distribution | Standard Deviation = 5 | 16.078507 seconds | 6.43×10-21% |
| Uniform Distribution | Standard Deviation = 10 | 1.569202 seconds | 5.01×10-25% |
| Normal Distribution | Standard Deviation = 1 | 0.941653 seconds | 6.11×10-28% |
| Normal Distribution | Standard Deviation = 5 | 0.891351 seconds | 1.28×10-28% |
| Normal Distribution | Standard Deviation = 10 | 1.464059 seconds | 1.54×10-28% |
| Sinusoidal | Frequency = 1 Hz, Amplitude = 1 | 10.395199 seconds | 39814% |
| Sinusoidal | Frequency = 10 Hz, Amplitude = 1 | 12.403733 seconds | 1993% |
| Sinusoidal | Frequency = 100 Hz, Amplitude = 1 | 5.686532 seconds | 46.47% |
| Sinusoidal | Frequency = 1 kHz, Amplitude = 1 | 11.499738 seconds | 2582% |
| Sinusoidal | Frequency = 10 kHz, Amplitude = 1 | 11.721615 seconds | 41948% |
| Sinusoidal | Frequency = 1 Hz, Amplitude = 10 | 11.594711 seconds | 2272% |
| Sinusoidal | Frequency = 10 Hz, Amplitude = 10 | 11.043747 seconds | 398% |
| Sinusoidal | Frequency = 100 Hz, Amplitude = 10 | 8.798550 seconds | 960150% |
| Sinusoidal | Frequency = 1 kHz, Amplitude = 10 | 10.548092 seconds | 5965% |
| Sinusoidal | Frequency = 10 kHz, Amplitude = 10 | 2.166551 seconds | 2.65×108% |
| Triangular | Width = 2π, Amplitude = 1 | 4.902682 seconds | 1.77×10-19% |
| Triangular | Width = 0.1×2π, Amplitude = 1 | 2.264347 seconds | 1.81×10-24% |
| Triangular | Width = 0.01×2π, Amplitude = 1 | 14.424037 seconds | 6.05×10-18% |
| Triangular | Width = 2π, Amplitude = 10 | 5.365355 seconds | 1.84×10-18% |
| Triangular | Width = 0.1×2π, Amplitude = 10 | 9.810197 seconds | 2.37×10-18% |
| Triangular | Width = 0.01×2π, Amplitude = 10 | 1.229538 seconds | 1.07×10-27% |

### Varying Orders

Setting ,

|  |  |  |
| --- | --- | --- |
| Order | Computation Time | %MSE |
| 2 | 1.907617 seconds | 2.64×10-25% |
| 3 | 5.331890 seconds | 2.68×10-25% |
| 4 | 26.402480 seconds | 2.68×10-25% |
| 5 | 67.471670 seconds | 2.68×10-25% |

### Analysis

For a simple linear difference equation we may conclude that:

* %MSE was almost zero because the system model consists of polynomial terms which are candidates of the FOS algorithm.
* %MSE w.r.t to original input improves with the decrease of noise at the output.
* %MSE w.r.t to noisy output is very close to ideal %MSE, therefore the FOS algorithm removes out the effect of noise due to the averaging which appears in the algorithm.
* Generating input randomly using normal distribution seems to have better results than using uniform distribution.
* Sinusoidal input generated very high error rates probably because limited number of values of and are generated. Increasing the frequency of sinusoidal input reduced %MSE up to a certain limit and then increased again. This is probably due to the effect of aliasing which means that the same sequence of inputs are generated when frequency increases.
* Triangular input surprisingly showed low %MSE although they have the same repetitive nature of sinusoidal signals. Further investigation is needed to evaluate whether the FOS model concluded using a triangular input shall give good results for different types of input. The effect of aliasing also appears for triangular input.
* Increasing the maximum-cross order of candidates shall not improve the %MSE but rather only increase computation time because the original model consist only of 1st order terms.

## Non-Linear Difference Equation

% Complex Difference Equation

y = zeros(3000, 1);

a0 = 1;

b0 = 0.7;

b1 = 0.8;

c1 = 0.1;

c2 = 0.4;

c4 = 0.2;

for n = 6:3000

y(n) = a0 + b0\*x(n) + b1\*x(n-1) + c1\*x(n-1)\*y(n-1) - c2\*x(n)\*x(n-1)+ c4\*x(n-4)\*x(n-5);

end

**Total Elapsed Time** = 32.708566 seconds

**Chosen values of K & L**: ,

 

**%MSE** = 2.397×10-24 %

|  |  |  |
| --- | --- | --- |
| K | L | Computation Time |
| 10 | 12 | 3.087012 seconds |
| 3 | 15 | 34.769338 seconds |
| 12 | 8 | 4.524928 seconds |
| 10 | 12 | 4.338774 seconds |
| 10 | 19 | 6.453342 seconds |
| 8 | 19 | 5.733319 seconds |
| 6 | 5 | 1.272192 seconds |
| 14 | 6 | 3.317925 seconds |
| 8 | 3 | 1.227670 seconds |
| 7 | 10 | 2.901604 seconds |

### Noise Inserted

|  |  |
| --- | --- |
| P = 150  %MSE = 3.048% | Total Elapsed Time = 14.707611 seconds  %MSE w.r.t. noisy output = 65.51%  Ideal %MSE = 61.36% |
|  |  |
| P = 100  %MSE = 2.70% | Total Elapsed Time = 14.885791 seconds  %MSE w.r.t. noisy output = 55.39%  Ideal %MSE = 51.16% |
|  |  |
| P = 50  %MSE = 0.48% | Total Elapsed Time = 20.216341 seconds  %MSE w.r.t. noisy output = 36.60%  Ideal %MSE = 34.06% |
|  |  |
| P = 25  %MSE = 0.45% | Total Elapsed Time = 22.122402 seconds  %MSE w.r.t. noisy output = 22.38%  Ideal %MSE = 20.37% |
|  |  |

### Different Input Generations

Setting ,

|  |  |  |  |
| --- | --- | --- | --- |
| Input Type | Parameter | Computation Time | %MSE |
| Uniform Distribution | Standard Deviation = 1 | 1.404847 seconds | 6.56×10-25% |
| Uniform Distribution | Standard Deviation = 5 | 1.064718 seconds | 2.01×10-27% |
| Uniform Distribution | Standard Deviation = 10 | N/A[[1]](#footnote-1) | |
| Normal Distribution | Standard Deviation = 1 | 1.292744 seconds | 2.28×10-28% |
| Normal Distribution | Standard Deviation = 5 | 1.135890 seconds | 4.70×10-28% |
| Normal Distribution | Standard Deviation = 10 | 1.126074 seconds | 2.28×10-29% |
| Sinusoidal | Frequency = 1 Hz, Amplitude = 1 | 2.128849 seconds | 0.239% |
| Sinusoidal | Frequency = 10 Hz, Amplitude = 1 | 7.927907 seconds | 95.67% |
| Sinusoidal | Frequency = 100 Hz, Amplitude = 1 | 2.761670 seconds | 0.0046% |
| Sinusoidal | Frequency = 1 kHz, Amplitude = 1 | 23.944826 seconds | 1.72×105% |
| Sinusoidal | Frequency = 10 kHz, Amplitude = 1 | 2.219971 seconds | 0.92% |
| Sinusoidal | Frequency = 1 Hz, Amplitude = 10 | 1.313247 seconds | 0.49% |
| Sinusoidal | Frequency = 10 Hz, Amplitude = 10 | 1.833226 seconds | 0.81% |
| Sinusoidal | Frequency = 100 Hz, Amplitude = 10 | 1.631035 seconds | 0.01% |
| Sinusoidal | Frequency = 1 kHz, Amplitude = 10 | 1.319631 seconds | 2.76×10-7% |
| Sinusoidal | Frequency = 10 kHz, Amplitude = 10 | 0.955011 seconds | 2.04×10-8% |
| Triangular | Width = 2π, Amplitude = 1 | 1.598169 seconds | 3.98×10-26% |
| Triangular | Width = 0.1×2π, Amplitude = 1 | 19.331429 seconds | 1.06×10-12% |
| Triangular | Width = 0.01×2π, Amplitude = 1 | 1.560098 seconds | 6.19×10-26% |
| Triangular | Width = 2π, Amplitude = 10 | 11.058648 seconds | 2.85×10-15% |
| Triangular | Width = 0.1×2π, Amplitude = 10 | 1.312038 seconds | 7.62×10-28% |
| Triangular | Width = 0.01×2π, Amplitude = 10 | 19.017779 seconds | 1.55×10-17% |

### Varying Orders

Setting ,

|  |  |  |
| --- | --- | --- |
| Order | Computation Time | %MSE |
| 2 | 1.346171 seconds | 2.50×10-24% |
| 3 | 5.354506 seconds | 2.50×10-24% |
| 4 | 17.021489 seconds | 2.50×10-24% |
| 5 | 58.637289 seconds | 2.50×10-24% |

### Analysis

For a non-linear difference equation we may conclude that:

* %MSE was almost zero because the system model consists of polynomial terms which are candidates of the FOS algorithm.
* %MSE w.r.t to original input improves with the decrease of noise at the output.
* %MSE w.r.t to noisy output is very close to ideal %MSE, therefore the FOS algorithm removes out the effect of noise due to the averaging which appears in the algorithm.
* Generating input randomly using normal distribution seems to have better results than using uniform distribution.
* Generating input of uniform distribution with high standard deviation may result in high divergence due to the feedback effect (i.e. current output depends on previous outputs) of the model.
* Sinusoidal and triangular inputs resulted in better results than for linear difference equations. Further investigation is needed to evaluate this.
* Increasing the maximum-cross order of candidates shall not improve the %MSE but rather only increase computation time because the original model consist only of 2nd order terms.

## LNL Cascade Model

% LNL Cascade

t = 1:5;

g1 = exp(-t) + exp(-2\*t);

a1 = 0.5 + 2\*exp(-t);

k1 = 3\*exp(-t);

g2 = exp(-t) + 3\*exp(-2\*t);

a2 = 0.2 + 3\*exp(-t);

k2 = 0.1\*exp(-t) + 0.9\*exp(-2\*t);

y = lnl(g1, a1, k1, x) + lnl(g2, a2, k2, x);

**Total Elapsed Time** = 54.206952 seconds

**Chosen values of K & L**: ,

 

**%MSE** = 1.3856 %

|  |  |  |
| --- | --- | --- |
| K | L | Computation Time |
| 10 | 12 | 11.016234 seconds |
| 3 | 15 | 10.190549 seconds |
| 12 | 8 | 11.830497 seconds |
| 10 | 12 | 15.097112 seconds |
| 10 | 19 | 25.064469 seconds |
| 8 | 19 | 24.624407 seconds |
| 6 | 5 | 3.742241 seconds |
| 14 | 6 | 14.313596 seconds |
| 8 | 3 | 4.309165 seconds |
| 7 | 10 | 7.572814 seconds |

### Noise Inserted

|  |  |
| --- | --- |
| P = 150  %MSE = 3.46% | Total Elapsed Time = 28.024301 seconds  %MSE w.r.t. noisy output = 60.99%  Ideal %MSE = 62.91% |
|  |  |
| P = 100  %MSE = 2.95% | Total Elapsed Time = 89.228064 seconds  %MSE w.r.t. noisy output = 50.50%  Ideal %MSE = 52.48% |
|  |  |
| P = 50  %MSE = 2.33% | Total Elapsed Time = 33.752851 seconds  %MSE w.r.t. noisy output = 33.45%  Ideal %MSE = 34.88% |
|  |  |
| P = 25  %MSE = 2.22% | Total Elapsed Time = 29.121019 seconds  %MSE w.r.t. noisy output = 20.88%  Ideal %MSE = 20.78% |
|  |  |

### Different Input Generations

Setting ,

|  |  |  |  |
| --- | --- | --- | --- |
| Input Type | Parameter | Computation Time | %MSE |
| Uniform Distribution | Standard Deviation = 1 | 2.097548 seconds | 7.57% |
| Uniform Distribution | Standard Deviation = 5 | 2.768679 seconds | 8.91% |
| Uniform Distribution | Standard Deviation = 10 | 2.759290 seconds | 9.03% |
| Normal Distribution | Standard Deviation = 1 | 3.719084 seconds | 43.83% |
| Normal Distribution | Standard Deviation = 5 | 5.288471 seconds | 46.77% |
| Normal Distribution | Standard Deviation = 10 | 4.395752 seconds | 47.48% |
| Sinusoidal | Frequency = 1 Hz, Amplitude = 1 | 0.819467 seconds | 0.0025% |
| Sinusoidal | Frequency = 10 Hz, Amplitude = 1 | 1.084350 seconds | 1.52×10-5% |
| Sinusoidal | Frequency = 100 Hz, Amplitude = 1 | 1.339380 seconds | 1.69×10-6% |
| Sinusoidal | Frequency = 1 kHz, Amplitude = 1 | 1.762254 seconds | 1.439×10-10% |
| Sinusoidal | Frequency = 10 kHz, Amplitude = 1 | 2.536762 seconds | 7.21×10-11% |
| Sinusoidal | Frequency = 1 Hz, Amplitude = 10 | 1.005178 seconds | 1.30×10-5% |
| Sinusoidal | Frequency = 10 Hz, Amplitude = 10 | 1.099383 seconds | 2.76×10-6% |
| Sinusoidal | Frequency = 100 Hz, Amplitude = 10 | 2.392238 seconds | 7.11×10-11% |
| Sinusoidal | Frequency = 1 kHz, Amplitude = 10 | 2.336597 seconds | 7.00×10-11% |
| Sinusoidal | Frequency = 10 kHz, Amplitude = 10 | 2.097544 seconds | 2.84×10-11% |
| Triangular | Width = 2π, Amplitude = 1 | 12.876216 seconds | 5.77×10-4% |
| Triangular | Width = 0.1×2π, Amplitude = 1 | 15.964115 seconds | 0.05% |
| Triangular | Width = 0.01×2π, Amplitude = 1 | 4.545124 seconds | 0.19% |
| Triangular | Width = 2π, Amplitude = 10 | 14.295613 seconds | 3.64×10-5% |
| Triangular | Width = 0.1×2π, Amplitude = 10 | 9.972019 seconds | 0.093% |
| Triangular | Width = 0.01×2π, Amplitude = 10 | 11.586914 seconds | 0.50% |

### Varying Orders

Setting , (close to the values which we know previously).

|  |  |  |
| --- | --- | --- |
| Order | Computation Time | %MSE |
| 2 | 2.338557 seconds | 1.44% |
| 3 | 35.752206 seconds | 0.082% |
| 4 | 1096.8528 seconds | 9.56×10-4% |
| 5 | 23159.321 seconds | 1.46×10-7% |

### Analysis

For a LNL cascade model we may conclude that:

* %MSE was relatively high (more than 1%) because the system model consists of polynomial terms of higher order terms which were not candidates of the FOS algorithm when using maximum order of 2 only.
* %MSE w.r.t to original input improves with the decrease of noise at the output.
* %MSE w.r.t to noisy output is very close to ideal %MSE, therefore the FOS algorithm removes out the effect of noise due to the averaging which appears in the algorithm.
* Generating input randomly using normal distribution and uniform distributions with higher standard deviations increases the %MSE. Therefore, increasing the variance of the input makes the FOS algorithm more vulnerable to errors if it does not contain the required candidates.
* Sinusoidal and triangular inputs resulted in better results than random data. But this is probably due to the repetitive nature of the inputs and outputs and the resulting FOS models are likely to result in erroneous results for non-repetitive inputs.
* Increasing the maximum-cross order of candidates improves the %MSE to levels of almost zero since the LNL model contains terms of higher order than 2.

## Non-Polynomial Equation

% Non-Polynomial Equation

y = zeros(3\*N, 1);

for n = 3:3\*N

y(n) = sin(x(n-1))\*cos(x(n)) + exp(-3\*x(n))\*sqrt(abs(x(n))) + 0.1\*log(abs(y(n-2)+0.01))\*y(n-1);

end

**Total Elapsed Time** = 91.730930 seconds

**Chosen values of K & L**: ,

 

**%MSE** = 2.0276 %

|  |  |  |
| --- | --- | --- |
| K | L | Computation Time |
| 10 | 12 | 22.192264 seconds |
| 3 | 15 | 9.406721 seconds |
| 12 | 8 | 19.798458 seconds |
| 10 | 12 | 23.021391 seconds |
| 10 | 19 | 38.117116 seconds |
| 8 | 19 | 27.037631 seconds |
| 6 | 5 | 8.068722 seconds |
| 14 | 6 | 13.414465 seconds |
| 8 | 3 | 6.237757 seconds |
| 7 | 10 | 9.920265 seconds |

### Noise Inserted

|  |  |
| --- | --- |
| P = 150  %MSE = 6.81% | Total Elapsed Time = 17.353620 seconds  %MSE w.r.t. noisy output = 59.92%  Ideal %MSE = 50.81% |
|  |  |
| P = 100  %MSE = 3.48% | Total Elapsed Time = 19.811687 seconds  %MSE w.r.t. noisy output = 48.13%  Ideal %MSE = 48.99% |
|  |  |
| P = 50  %MSE = 2.85% | Total Elapsed Time = 26.531140 seconds  %MSE w.r.t. noisy output = 32.33%  Ideal %MSE = 32.70% |
|  |  |
| P = 25  %MSE = 2.63% | Total Elapsed Time = 25.260854 seconds  %MSE w.r.t. noisy output = 20.29%  Ideal %MSE = 19.67% |
|  |  |

### Different Input Generations

Setting ,

|  |  |  |  |
| --- | --- | --- | --- |
| Input Type | Parameter | Computation Time | %MSE |
| Uniform Distribution | Standard Deviation = 1 | 2.942042 seconds | 20.56% |
| Uniform Distribution | Standard Deviation = 5 | 0.563647 seconds | 96.62% |
| Uniform Distribution | Standard Deviation = 10 | 0.704365 seconds | 94.37% |
| Normal Distribution | Standard Deviation = 1 | 4.095902 seconds | 58.75% |
| Normal Distribution | Standard Deviation = 5 | N/A[[2]](#footnote-2) | |
| Normal Distribution | Standard Deviation = 10 |
| Sinusoidal | Frequency = 1 Hz, Amplitude = 1 | 19.752543 seconds | 1.29% |
| Sinusoidal | Frequency = 10 Hz, Amplitude = 1 | 15.155718 seconds | 2.13% |
| Sinusoidal | Frequency = 100 Hz, Amplitude = 1 | 24.461126 seconds | 2.23% |
| Sinusoidal | Frequency = 1 kHz, Amplitude = 1 | 5.441387 seconds | 0.07% |
| Sinusoidal | Frequency = 10 kHz, Amplitude = 1 | 14.903009 seconds | 2.26% |
| Sinusoidal | Frequency = 1 Hz, Amplitude = 10 | 18.959170 seconds | 1.29% |
| Sinusoidal | Frequency = 10 Hz, Amplitude = 10 | 14.603828 seconds | 2.13% |
| Sinusoidal | Frequency = 100 Hz, Amplitude = 10 | 24.214600 seconds | 2.23% |
| Sinusoidal | Frequency = 1 kHz, Amplitude = 10 | 5.369926 seconds | 0.068% |
| Sinusoidal | Frequency = 10 kHz, Amplitude = 10 | 18.129356 seconds | 2.21% |
| Triangular | Width = 2π, Amplitude = 1 | 12.826004 seconds | 0.0032% |
| Triangular | Width = 0.1×2π, Amplitude = 1 | 5.200054 seconds | 0.187% |
| Triangular | Width = 0.01×2π, Amplitude = 1 | 4.808909 seconds | 0.385% |
| Triangular | Width = 2π, Amplitude = 10 | N/A[[3]](#footnote-3) | |
| Triangular | Width = 0.1×2π, Amplitude = 10 |
| Triangular | Width = 0.01×2π, Amplitude = 10 |

### Varying Orders

Setting ,

|  |  |  |
| --- | --- | --- |
| Order | Computation Time | %MSE |
| 2 | 6.928569 seconds | 1.41% |
| 3 | 33.842255 seconds | 0.56% |
| 4 | 447.53317 seconds | 0.22% |
| 5 | 4475.6902 seconds | 0.106% |

### Analysis

For a non-polynomial model we may conclude that:

* %MSE was relatively high (more than 2%) because the system model may be expanded using Taylor series to higher order terms which were not candidates of the FOS algorithm when using maximum order of 2 only.
* %MSE w.r.t to original input improves with the decrease of noise at the output.
* %MSE w.r.t to noisy output is very close to ideal %MSE, therefore the FOS algorithm removes out the effect of noise due to the averaging which appears in the algorithm.
* Generating input randomly using normal distribution and uniform distributions with higher standard deviations increases the %MSE. Therefore, increasing the variance of the input makes the FOS algorithm more vulnerable to errors if it does not contain the required candidates.
* Sinusoidal and triangular inputs resulted in better results than random data. But this is probably due to the repetitive nature of the inputs and outputs and the resulting FOS models are likely to result in erroneous results for non-repetitive inputs.
* The higher the maximum order of the cross-product terms of the FOS algorithm, the lower %MSE we get. However, we don’t reach the almost-zero %MSE we got in previous models, because the non-polynomial model is equivalent to a model of infinite order.

## Static Gyroscope Data

A gyroscope is a device rate of angular rotation. I used readings from a stationary MEMS grade gyroscope. When stationary, its reading should be zero. However, there is a usually white noise as well as bias error which drifts with time. I here try to model this bias drift, which we labeled as , with time, which we labeled as . I start with removing noise using wavelet denoising.

% Gyroscope Data

load wz.mat;

wzd = wden(wz, 'heursure', 's', 'one', 15, 'db4');

x = (1:3\*N)';

y = wzd(1:3\*N);

**Total Elapsed Time** = 35.985119 seconds

**Chosen values of K & L**: ,

 

We also plot the auto-correlation of the output using the following code:

figure(3);

Rw = xcorr(w - mean(w));

plot(-length(Rw)/2:length(Rw)/2-1 , Rw);



**%MSE** = 1.897×10-5%

|  |  |  |
| --- | --- | --- |
| K | L | Computation Time |
| 10 | 12 | 3.913472 seconds |
| 3 | 15 | 4.546578 seconds |
| 12 | 8 | 3.950725 seconds |
| 10 | 12 | 5.179663 seconds |
| 10 | 19 | 5.546014 seconds |
| 8 | 19 | 4.829279 seconds |
| 6 | 5 | 0.691572 seconds |
| 14 | 6 | 2.847456 seconds |
| 8 | 3 | 0.748498 seconds |
| 7 | 10 | 2.982398 seconds |

### Reduce De-Noising

|  |  |
| --- | --- |
| Wavelet Scale = 3, P = 2872  %MSE = 15846% | Total Elapsed Time = 123.977470 seconds  %MSE w.r.t. noisy output = 0.39%  Ideal %MSE = 102.5% |
|  |  |
|  | |
| Wavelet Scale = 6, P = 278.1  %MSE = 3288% | Total Elapsed Time = 72.600251 seconds  %MSE w.r.t. noisy output = 8.86×10-4%  Ideal %MSE = 121.5% |
|  |  |
|  | |
| Wavelet Scale = 8, P = 113.08  %MSE = 2032% | Total Elapsed Time = 42.758778 seconds  %MSE w.r.t. noisy output = 6.98×10-6%  Ideal %MSE = 76.1% |
|  |  |
|  | |
| Wavelet Scale = 12, P = 3.38  %MSE = 29.03% | Total Elapsed Time = 50.976234 seconds  %MSE w.r.t. noisy output = 2.68×10-4%  Ideal %MSE = 2.52% |
|  |  |
|  | |
| Wavelet Scale = 13, P = 0.59  %MSE = 35.48% | Total Elapsed Time = 57.308801 seconds  %MSE w.r.t. noisy output = 2.21×10-7%  Ideal %MSE = 0.517% |
|  |  |
|  | |
| Wavelet Scale = 14, P = 0.0256  %MSE = 6.6% | Total Elapsed Time = 47.901612 seconds  %MSE w.r.t. noisy output = 1.58×10-4%  Ideal %MSE = 0.0249% |
|  |  |
|  | |

### Different Input Generations

This test is not applicable because the input here is time which has a constant representation and cannot be generated differently.

### Varying Orders

Setting ,

|  |  |  |
| --- | --- | --- |
| Order | Computation Time | %MSE |
| 2 | 0.617821 seconds | 1.92×10-7% |
| 3 | 3.627746 seconds | 1.92×10-7% |
| 4 | 13.204463 seconds | 1.92×10-7% |
| 5 | 44.273138 seconds | 1.92×10-7% |

### Analysis

For modeling a gyroscope we may conclude that:

* %MSE was relatively low and therefore it seems that the cross-product terms of order 2 containing and are enough.
* %MSE w.r.t to original input improves with the increase of denoising. However, the results are not similar as to noise additions shown in previous examples because the noise which is being removed by wavelet denoising are probably not zero-mean white noise.
* Increasing the maximum order of cross-product terms did not improve %MSE and therefore it seems that the cross-product terms of order 2 containing and are enough.

## Static Accelerometer Data

% Gyroscope Data

load 'F:\EE517 - Project 3 Data\Xbow\_stat\_data\_other.mat';

x = (1:3\*N)';

y = f.x(1:3\*N);

clear f w interp\_info denoising\_info denoising\_info\_ orig\_data\_info;

**Total Elapsed Time** = 31.910843 seconds

**Chosen values of K & L**: ,

 

We also plot the auto-correlation of the output using the following code:

figure(3);

Rw = xcorr(w - mean(w));

plot(-length(Rw)/2:length(Rw)/2-1 , Rw);



**%MSE** = 0.0017%

|  |  |  |
| --- | --- | --- |
| K | L | Computation Time |
| 10 | 12 | 2.519522 seconds |
| 3 | 15 | 3.236106 seconds |
| 12 | 8 | 2.509121 seconds |
| 10 | 12 | 3.290113 seconds |
| 10 | 19 | 7.709763 seconds |
| 8 | 19 | 6.706057 seconds |
| 6 | 5 | 0.610797 seconds |
| 14 | 6 | 1.822470 seconds |
| 8 | 3 | 0.613365 seconds |
| 7 | 10 | 2.087462 seconds |

### Noise Inserted

|  |  |
| --- | --- |
| P = 150  %MSE = 22.31% | Total Elapsed Time = 15.706235 seconds  %MSE w.r.t. noisy output = 69.26%  Ideal %MSE = 60.77% |
|  |  |
|  | |
| P = 100  %MSE = 16.72% | Total Elapsed Time = 16.757993 seconds  %MSE w.r.t. noisy output = 58.86%  Ideal %MSE = 50.65% |
|  |  |
|  | |
| P = 50  %MSE = 9.66% | Total Elapsed Time = 16.855352 seconds  %MSE w.r.t. noisy output = 40.25%  Ideal %MSE = 33.74% |
|  |  |
|  | |
| P = 25  %MSE = 5.88% | Total Elapsed Time = 16.007723 seconds  %MSE w.r.t. noisy output = 25.03%  Ideal %MSE = 20.21% |
|  |  |
|  | |

### Different Input Generations

This test is not applicable because the input here is time which has a constant representation and cannot be generated differently.

### Varying Orders

Setting ,

|  |  |  |
| --- | --- | --- |
| Order | Computation Time | %MSE |
| 2 | 2.757120 seconds | 0.0043% |
| 3 | 18.079261 seconds | 0.0043% |
| 4 | 110.92006 seconds | 0.0043% |
| 5 | 508.46559 seconds | 0.0043% |

### Analysis

For modeling an accelerometer we may conclude that:

* The higher peak of the auto-correlation function around zero value, means that there is correlation between the outputs of the accelerometer and therefore there is more randomness in the output which cannot be modeled by FOS.
* %MSE was relatively low and therefore it seems that the cross-product terms of order 2 containing and are enough.
* %MSE is much higher when noise is inserted when compared with previous models. This is probably due to the fact that the model of the accelerometer is partially stochastic and cannot be totally modeled deterministically.
* Increasing the maximum order of cross-product terms did not improve %MSE and therefore it seems that the cross-product terms of order 2 containing and are enough.

# Conclusion

The FOS algorithm proved to be excellent in modeling deterministic models resulting in %MSE which are almost zeros. However, some real-life models seem to have some stochastic nature and therefore the accuracy of FOS reduces to 1-3%. For complex deterministic models, FOS results in better results the higher the maximum cross-order and lags of its candidates. However, a balance has to be made between seeking higher accuracy and seeking less computation time.

Normal distribution followed by uniform distribution of random inputs seems to result in highest accuracies. They are better than periodic inputs since they cover more combination of values and more combination of orders of values due to their random nature.

# Code

## Main Code

Training Phase

|  |
| --- |
| %% Training Phase  tic  % Apply FOS for 1st 1000 samples of data  rng(2);  K = randi([3,20], 10, 1);  rng(3);  L = randi([3,20], 10, 1);  for i = 1:length(K)  order = 2;  N0 = max(K(i),L(i));  %tic  [at{i}, pt{i}] = fos( x(1:N), w(1:N), K(i), L(i), order );  %toc    y1 = evalfunct( x(1:N), w(1:N), pt{i}, at{i} );  e = y1 - w(1:N);  MSEpercent(i) = mean(e(N0+1:N).^2)/var(w(N0+1:N)) \*100;  end |

Selection Phase

|  |
| --- |
| %% Selection Phase  clear MSEpercent;  % Apply FOS for 2nd 1000 samples of data  for i = 1:1:length(K)  y1 = evalfunct( x(N+1:2\*N), w(N+1:2\*N), pt{i}, at{i} );  e = y1 - w(N+1:2\*N);  MSEpercent(i) = mean(e(N0+1:N).^2)/var(w(N+N0+1:2\*N)) \*100;  end    % Choose the best model over the 2nd 1000 samples  index = find(MSEpercent == min(MSEpercent));  as = at{index(1)};  ps = pt{index(1)}; |

Evaluation Phase

|  |
| --- |
| %% Evaluation Phase  clear MSEpercent;  % Apply FOS for 3nd 1000 samples of data  y1 = evalfunct( x(2\*N+1:3\*N), w(2\*N+1:3\*N), ps, as);  e = y1 - y(2\*N+1:3\*N);  MSEpercent = mean(e(N0+1:N).^2)/var(y(2\*N+N0+1:3\*N)) \*100;  IdealMSEPercent = var(r) / var(w) \* 100;    e = y1 - w(2\*N+1:3\*N);  MSEpercent1 = mean(e(N0+1:N).^2)/var(w(2\*N+N0+1:3\*N)) \*100; |

Plot Results

|  |
| --- |
| %% Plot  y1 = evalfunct( x, w, ps, as);  figure(1);  plot(y, 'b'); hold on;  plot(N0+1:3\*N, y1(N0+1:3\*N),'r');  xlabel('n');  ylabel('y[n]');  legend('y', 'y1');    figure(3);  Rw = xcorr(w - mean(w));  plot(-length(Rw)/2:length(Rw)/2-1 , Rw);  title('Auto-correlation of output w[n]');  xlabel('n');  ylabel('R(w)');    toc    %% Plot Q[m] for best K & L  order = 2;  N0 = max( K(index(1)) , L(index(1)) );  [a, p] = fos( x(1:N), y(1:N), K(index(1)), L(index(1)), order ); |

## Input Generation

Default generation

|  |
| --- |
| % Uniformly distributed pseudorandom numbers [0,1]  rng(0);  x = rand(3\*N, 1); |

Uniform Distribution

|  |
| --- |
| % Uniform distribution  sigmax = 1;  rng(0);  x1 = rand(3\*N, 1);  x = x1 / std(x1) \* sigmax; |

Normal Distribution

|  |
| --- |
| % Normal distribution  sigmax = 1;  rng(0);  x1 = randn(3\*N, 1);  x = x1 / std(x1) \* sigmax; |

Sinusoidal Input

|  |
| --- |
| % Sinusoidal  f = 1;  A = 1;  t = (1:3\*N)';  x = A\*sin(2\*pi\*f\*t); |

Triangular Input

|  |
| --- |
| % Triangular  width = 1;  A = 1;  t = (1:3\*N)';  x = A\*sawtooth(t,width); |

## System Models

Linear Difference Equation

|  |
| --- |
| % Simple Difference Equation 1  y = 1 + 0.6\*x + 0.3\*delay(x,1) + 0.4\*delay(x,2) + 0.7\*delay(x,3); |

Complex Difference Equation

|  |
| --- |
| % Complex Difference Equation  y = zeros(3\*N, 1);    a0 = 1;  b0 = 0.7;  b1 = 0.8;  c1 = 0.1;  c2 = 0.4;  c4 = 0.2;    for n = 6:3\*N  y(n) = a0 + b0\*x(n) + b1\*x(n-1) + c1\*x(n-1)\*y(n-1) - c2\*x(n)\*x(n-1)+ c4\*x(n-4)\*x(n-5);  end |

LNL Cascade

|  |
| --- |
| % LNL Cascade  i = 1:5;  g1 = exp(-i) + exp(-2\*i);  a1 = 0.5 + 2\*exp(-i);  k1 = 3\*exp(-i);    g2 = exp(-i) + 3\*exp(-2\*i);  a2 = 0.2 + 3\*exp(-i);  k2 = 0.1\*exp(-i) + 0.9\*exp(-2\*i);    y = lnl(g1, a1, k1, x) + lnl(g2, a2, k2, x); |

Non-Polynomial Equation

|  |
| --- |
| % Non-Polynomial Equation  y = zeros(3\*N, 1);  for n = 3:3\*N  y(n) = sin(x(n-1))\*cos(x(n)) + exp(-3\*x(n))\*sqrt(abs(x(n))) + 0.1\*log(abs(y(n-2)+0.01))\*y(n-1);  end |

Gyroscope Data

|  |
| --- |
| % Real Data 1 - Project 2  load wz.mat;  wzd = wden(wz, 'heursure', 's', 'one', 15, 'db4');  x = (1:3\*N)';  y = wzd(1:3\*N);  w = wden(wz, 'heursure', 's', 'one', 14, 'db4'); w=w(1:length(x));  r = w - y;  P = 100 \* var(r)/var(y) |

Accelerometer Data

|  |
| --- |
| % Real Data 2 - Project 3  load 'C:\Data Logging\EE 517 Winter 2012\Project 3\Xbow\_stat\_data\_other.mat';  x = (1:3\*N)';  y = f.x(1:3\*N);  clear f w interp\_info denoising\_info denoising\_info\_ orig\_data\_info; |

Noise Insertion

|  |
| --- |
| % Noise Insertion  rng(1);  r1 = wgn(3\*N, 1, 0);  r1 = r1 - mean(r1);  r = r1 / std(r1) \* sqrt(P/100) \* std(y);  w = r + y; |

## FOS Algorithm

|  |
| --- |
| function [ a, p ] = fos( x, y, K, L, order )  %FOS Summary of this function goes here  % Detailed explanation goes here  N = length(x);  N0 = max(K,L);    h = waitbar(0,'1','Name','FOS Calculation...',...  'CreateCancelBtn',...  'setappdata(gcbf,''canceling'',1)');  setappdata(h,'canceling',0)  % Structure of p:  % p.x = delays of different x terms.  % p.y = delays of different y terms.  % p.x + p.y <= order  p = struct('const', 1, 'x', [], 'y', []);  P = struct(p);    g(0 +1) = mean(y(N0+1:N));  D(0 +1,0 +1) = 1;  C(0 +1) = mean(y(N0+1:N));  P(0 +1) = [];  Q(0 +1) = g(0 +1)^2 \* D(0 +1,0 +1);    waitbar(0, h,'Generating Candidates...');    % generate all candidates  i = 1;  for torder = 1 : order  waitbar(torder / order, h);  for xorder = 0:torder  if getappdata(h,'canceling')  delete(h);  return;  end    yorder = torder - xorder;    xdelays = combsrep(0:K, xorder);  ydelays = combsrep(1:L, yorder);    if (size(xdelays,1) >= 1)  for j = 1:size(xdelays,1)  P(i).x = xdelays(j, :);  if (size(ydelays,1) >= 1)  for k = 1:size(ydelays,1)  P(i).y = ydelays(k, :);    i = i + 1;  end  else  i = i+1;  end  end  else  for k = 1:size(ydelays,1)  P(i).y = ydelays(k, :);    i = i + 1;  end  end  end  end    waitbar(0, h, 'Evaluating Candidates...');    M = 1;  while (true)  if getappdata(h,'canceling')  delete(h);  return;  end    waitbar(0, h, sprintf('Evaluating Candidate %d...', M));  m = M;    % Evaluate Q for each candidate  clear Qc;  if (isempty(P))  break;  end  for i=1:length(P)  if getappdata(h,'canceling')  delete(h);  return;  end    waitbar(i / length(P), h);  Pval = evalterm(x, y, P(i));    D(m+1,1) = mean(Pval(N0+1:N));  for j=1:m  if getappdata(h,'canceling')  delete(h);  return;  end    alpha(m+1, j) = D(m+1, j) ./ D(j, j);  if (j < M)  pval = evalterm(x, y, p(j+1));  else  pval = Pval;  end  D(m+1, j+1) = mean(Pval(N0+1:N) .\* pval(N0+1:N)) - sum(alpha(j+1, 1:j) .\* D(m+1, 1:j));  end  C(m+1) = mean(y(N0+1:N) .\* Pval(N0+1:N)) - sum(alpha(m+1, 1:m) .\* C(1:m));    g(m+1) = C(m+1)/D(m+1, m+1);  Qc(i) = g(m+1)^2 \* D(m+1, m+1);  end    % Find index of maximum Q  index = find(Qc == max(Qc));  Pval = evalterm(x, y, P(index(1)));    for j=1:m  if getappdata(h,'canceling')  delete(h);  return;  end    D(m+1,1) = mean(Pval(N0+1:N));  for j=1:m  if getappdata(h,'canceling')  delete(h);  return;  end    alpha(m+1, j) = D(m+1, j) ./ D(j, j);  if (j < M)  pval = evalterm(x, y, p(j+1));  else  pval = Pval;  end  D(m+1, j+1) = mean(Pval(N0+1:N) .\* pval(N0+1:N)) - sum(alpha(j+1, 1:j) .\* D(m+1, 1:j));  end    end  C(m+1) = mean(y(N0+1:N) .\* Pval(N0+1:N)) - sum(alpha(m+1, 1:m) .\* C(1:m));  Q(m+1) = max(Qc);    diagD = diag(D)';  if ( Q(M+1) < 4/(N - N0 + 1) \*(mean(y(N0+1:N).^2) - sum(Q(1:M))))  M = M - 1;  break;  end    p(m+1) = P(index(1));  P(index(1)) = []; % remove it from the P    % find the coefficient of the chosen candidate  g(m+1) = C(m+1) / D(m+1,m+1);    if (isempty(P) || Q(m+1)<1e-26 )  break;  end    M = M + 1;  end    figure(2);  plot(0:length(Q)-1, Q(1:end));  ylabel('Q[m]');  xlabel('m');    % Obtain a  for i=0:M  v(i +1)=1;  for j = i+1:M  v(j +1) = -sum(alpha(j +1,i +1 : j-1 +1) .\* v(i +1 : j-1 +1));  end  a(i +1) = sum(g(i +1:M +1).\*v(i +1:M +1));  end    delete(h);    end |

## Helper Functions

Function to evaluate term.

|  |
| --- |
| function [ p ] = evalterm( x, y, lags)  %GETFUNCTION Summary of this function goes here  % Detailed explanation goes here    p = ones(length(x),1);    for i=1:length(lags.x)  k = lags.x(i);  p = p.\*delay(x,k);  end    for i=1:length(lags.y)  l = lags.y(i);  p = p.\*delay(y,l);  end    end |

Function to evaluate function.

|  |
| --- |
| function [ y1 ] = evalfunct( x, y, p, a )  %EVALFUNCT Summary of this function goes here  % Detailed explanation goes here    y1 = ones(length(x),1);  y1 = y1 \* a(1);  for j=2:length(p)  y1 = y1 + a(j) \* evalterm(x, y, p(j));  end    end |

Function to delay a signal.

|  |
| --- |
| function [ x1 ] = delay( x, k )  %DELAY Summary of this function goes here  % Detailed explanation goes here    x1 = [zeros(k,1); x(1:end-k)];    end |

Function to implement LNL system.

|  |
| --- |
| function [y, u] = lnl( g, a, k, x )  %LNL Summary of this function goes here  % Detailed explanation goes here  u = filter(1, g, x);    v = zeros(size(u));  for i=1:length(a)  v = v + a(i)\*u.^i;  end    y = filter(1, k, v);    end |

1. diverged to [↑](#footnote-ref-1)
2. diverged to [↑](#footnote-ref-2)
3. diverged to [↑](#footnote-ref-3)