AE4320 Assignment: Neural Networks

This individual take-home assignment consists of 4 parts. The deliverable of the assignments are a report of **maximum 25 pages** containing the theory and results of the 4 parts, and a set of Matlab or Python code files which were used to generate the results. Use the structure in this assignment (i.e. Part 1, Part 2, etc) to structure your report.
A total of 100 points can be obtained for the assignment. In **Appendix A**, a brief introduction to the measurement dataset and special validation dataset is given. In **Appendix B** details for the construction of the Kalman Filter are given. In **Appendix C**, a brief description of the supplied Matlab files is given. In **Appendix D** a description of the neural network structures is given.

Important: your report should have the same organization (and chapter numbering) as the assignment text!

Part 1: State & Parameter estimation with F-16 flight data (35 points)

You are asked first to implement a state estimation routine (Kalman Filter) to estimate a bias in the angle of attack measurements in the given dataset.

It is given that there is an unknown bias in the angle of attack (α_m) measurements caused by upwash. Also, a sensor glitch has been reported that may have corrupted part of the C_m measurements. The angle of sideslip (β_m) measurements and velocity (V_m) are not biased. Please see **Appendix B** for details on the construction of the Kalman Filter. **The report should contain the following items:**

1. Explain how you treated the given dataset, e.g. did you pre-filter the data, remove glitches, how did you define identification and validation datasets? (**6 points**)
2. Formulate the (trivial) system equations $\dot{x}_k = f(x_k, u_k, t)$ and the (non-trivial) output equations $z = h(x_k, u_k, t)$ given the information in **Appendix B**. (**3 points**)
3. Briefly motivate which type of Kalman filter should be used in this case; the linear Kalman Filter, the Extended Kalman Filter, or the Iterated Extended Kalman Filter. (**2 points**)
4. Construct your Kalman filter, and use it to estimate $C_{\alpha_{up}}$ (clearly present the numerical result!). **Important:** Use a fixed time step of $\Delta t = 0.01$ in your integrator/continuous-to-discrete time conversion operation, otherwise the input vector cannot be used correctly. Prove that your filter has converged. (**5 points**).
5. Reconstruct α_{true} using the estimated $C_{\alpha_{up}}$, and clearly show the difference with the measured α_m. **(2 points)**

Implement an ordinary least squares estimator for a simple polynomial model structure for the reconstructed F-16 flight data and apply it on the given dataset. The report/code should contain the following items:

6. Formulate a least squares estimator, and estimate the coefficients of your polynomial model. (**5 points**)
7. Show the influence of polynomial model order on the accuracy of fit. Do this for both the measurement dataset as well as the 'special validation dataset' that was included in the assignment. (**5 points**)
8. Show the results from the model validation; perform both a statistical and a model-error based validation. Do this for the measurement dataset (**not** the 'special validation dataset') that was included in the assignment. (**5 points**)

9. Provide the residual RMS of your selected polynomial model when validated using the special validation dataset that was included in the assignment. **Note** that the special validation dataset domain not necessarily overlaps the measurement data domain! (**2 points**)
10. Hand in the code used in this part in a separate, well documented file.

Part 2: Create a Radial Basis Function Neural Network (25 points)

You are asked to implement a feed-forward and radial basis function neural network

1. Program a Radial Basis Function (RBF) neural network using the format provided in **Appendix D**. Use the following network activation function:

$$\phi_j(v_j) = a \cdot \exp(-v_j); \quad v_j = \sum_i w_{ij}^2 (x_i - c_{ij})^2$$

   Approximate the reconstructed F-16 dataset using a linear regression approach to the neural networks. Clearly show in your report what the structure is of your neural network (number and placing of RBFs), and which design choices you made. **Hint**: in this case only the RBF amplitude (variable 'a') is a optimization parameter. (**4 points**)
2. Program a Levenberg-Marquardt (LM) learning algorithm for the RBF neural net. The algorithm can be found in the lecture slides. First clearly show how you formulated the weight vector **w**, the error vector **e**, and the Jacobian **J**. (**5 points**)
3. Approximate the reconstructed F-16 dataset using the LM algorithm that optimizes not only the RBF amplitude, but also RBF 'width' and location. How did you tune the damping variable? Clearly indicate in your report what the advantages and disadvantages are of the LM approach compared to that of part 3.1 in terms of sensitivity to initial conditions and convergence speed. (**5 points**)
4. Clearly demonstrate the effect the damping variable μ has on network convergence. How does a variable damping rate affect the results? (**5 point**)
5. Design an optimization algorithm that optimizes an RBF neural network in terms total number of neurons. Clearly explain your algorithm in the report. Show that the approximation power of your optimized neural network exceeds that of the non-optimized neural network (**4 points**).
6. Provide the residual RMS of your 'best' RBF network model when validated using the **special validation dataset** that was included in the assignment. **Note** that the special validation dataset domain not necessarily overlaps the measurement data domain! (**2 points**)
7. Hand in the code used in this part in a separate, well documented file.

Part 3: Create a Feed-Forward Neural Network model (30 points)

You are asked to implement a feed-forward and radial basis function neural network

1. Program a Feed-Forward (FF) neural network using the format provided in **Appendix D**. Approximate the reconstructed F-16 dataset using a back-propagation algorithm. Use the algorithm found in the lecture slides. Use the tangent sigmoidal function as activation function:

$$\phi_j(v_j) = \frac{2}{1+\exp(-2v_j)} - 1$$

   The activation function 'purelin' is simple: y = v.
   Clearly show in your report what the structure is of your neural network (number of neurons), and which design choices you made. Also analyse the sensitivity of the back propagation approach to initial conditions (**4 points**)
2. Use the Levenberg-Marquardt learning algorithm presented in the lecture slides to train a FF neural network, and compare the results with the back-propagation algorithms in terms of approximation accuracy, required number of iterations, and sensitivity to initial conditions. Clearly report your findings in the report. Clearly show how you formulated the weight vector **w**, the error vector **e**, and the Jacobian **J**. (**10 points**)
3. Clearly demonstrate the effect the learning rate parameter η has on network convergence. How does a variable learning rate affect the results? (**4 point**)
4. Program an optimization algorithm that optimizes an FF neural network in terms total number of neurons. (**3 points**)
5. Compare the approximation accuracy of the FF neural network to that of an RBF neural network with the same number of parameters using the reconstructed F-16 data. Compare the total number of required learning iterations for both neural network types to reach a certain level of accuracy. Finally, compare the results with that of the polynomial model you identified in part 2. Clearly present and explain the results in your report. (**5 points**).
6. Provide the residual RMS of your selected simplex polynomial model when validated using the special validation dataset that was included in the assignment. **Note** that the special validation dataset domain not necessarily overlaps the measurement data domain! Compare with the results from **part 2** and **part 3** (**4 points**)
7. Hand in the code used in this part in a separate, well documented file.

Part 4: Conclusion (10 points)

You are asked to write a brief conclusion (max 1 page) for your report which should contain the following items:

1. Compare the numerical results from the modelling with the standard polynomial model, the RBF neural network models (including the linear regression variant) and the FF neural network model with each other. Provide a numerical comparison in a clear table in which the most important validation metrics are presented. (**6 points**)
2. Create a table with the strengths and weakness of each approach. (**4 points**)

Appendix A: Introduction to the F16 dataset.

Measurement Dataset

The file 'F16traindata_CMabV_2024.mat' contains the data in the Matlab MAT format , 'F16traindata_CMabV_2024.csv' contains the data in CSV format, which is more useful when using e.g. Python. The data is loaded from disk with Matlab using the command:

```
load('F16traindata_CMabV_2024.mat', 'Cm', 'Z_k', 'U_k');
```

The vector 'Cm' contains the measurement data of the pitching moment coefficient: $C_m \in \mathbb{R}^{N\times 1}$ with 'N' the total number of measurements. The matrix 'Z_k' contains the measurement locations (states), where Z_k contains the following data: $Z_k = \begin{bmatrix} \alpha_m & \beta_m & V_m \end{bmatrix} \in \mathbb{R}^{N\times 3}$. The matrix 'U_k' contains perfect linear accelerometer measurements that can be used in your Kalman filter. It contains the following data: $U_k = \begin{bmatrix} \dot{u} & \dot{v} & \dot{w} \end{bmatrix} \in \mathbb{R}^{N\times 3}$, with $\dot{u}, \dot{v}, \dot{w}$ the perfect (no biases, noise-free) time derivatives of the body velocities u, v, w. **Please do not confuse the longitudinal velocity '** u **' with the system input** U_k **!**

When plotting the data, which can be done with the supplied file 'F16_PlotData.m' you will find that α_m, β_m, V_m are contaminated with zero mean white noise. In addition α_m is biased due to upwash at the location of the angle of attack vane, see **Appendix B**. The measurements made on Cm are also contaminated with noise.

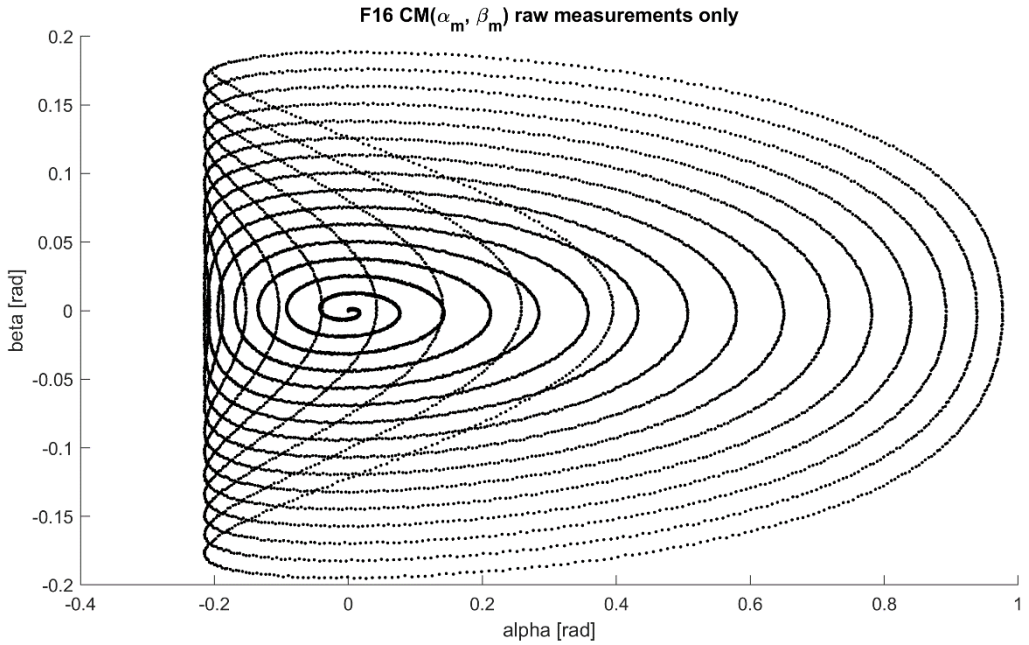

Figure 1: Locations in the angle of attack and angle of sideslip domain at which measurements of Cm are available.

Special validation dataset

This year, we have a **special validation dataset** which contains noise-free data on a very few select alpha,beta points. **These points do not need to be processed with your Kalman filter!**

The file ‘F16validationdata_2024.mat’ contains the **special validation dataset** in the Matlab MAT format , ‘F16validationdata _CMab_2024.csv’ contains the data in CSV format, which is more useful when using e.g. Python. The special validation dataset is loaded from disk with Matlab using the command:

```
load('F16validationdata_2024.mat', 'Cm_val', 'alpha_val', 'beta_val');
```

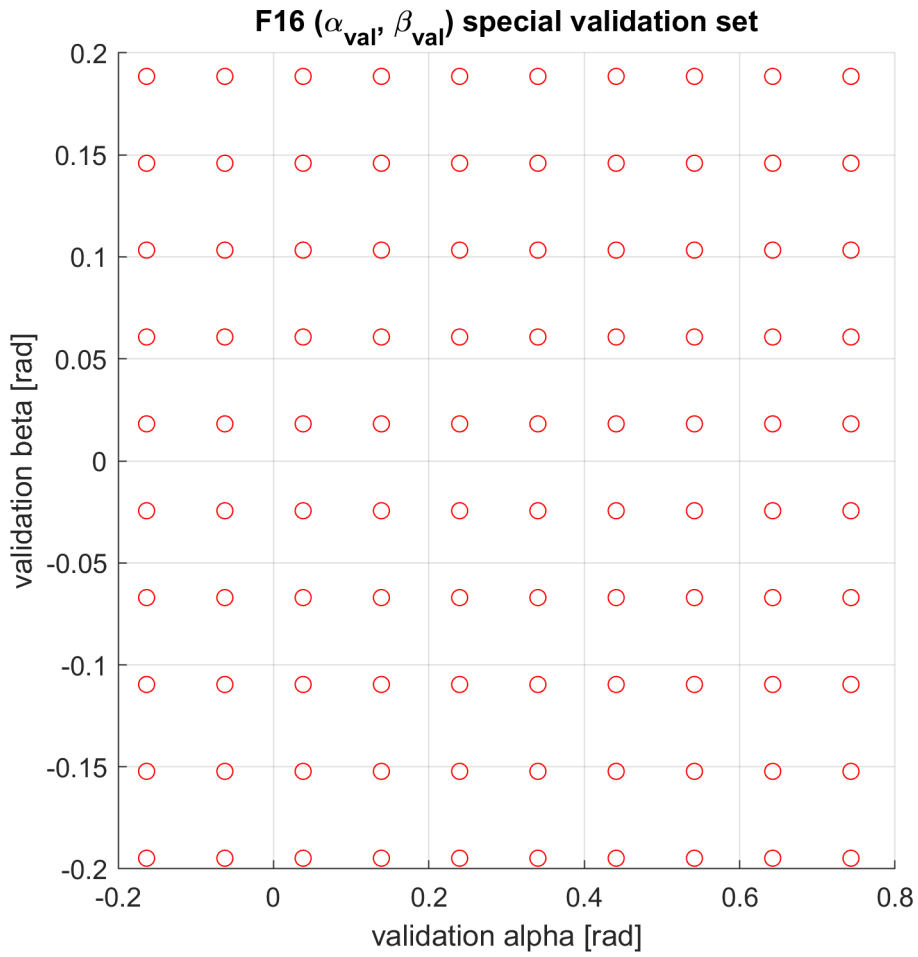

Figure 2: special validation dataset consisting of 100 points laid out on rectangular grid

Appendix B: Constructing the Kalman filter

In the construction of your Kalman filter, use the following state vector:

$$x_k = \begin{bmatrix} u & v & w & C_{\alpha_{up}} \end{bmatrix}^T$$

As measurement vector use:

$$z_k = \begin{bmatrix} \alpha_m & \beta_m & V_m \end{bmatrix}^T$$

As input vector use the perfect (**unbiased, noise-free**) linear accelerometer values, which are sampled with $\Delta t = 0.01$:

$$u_k = \begin{bmatrix} \dot{u} & \dot{v} & \dot{w} \end{bmatrix}$$

It is given that there is an unknown bias in the angle of attack (α_m) measurements caused by upwash at the location of the angle of attack vane. The angle of sideslip (β_m) measurements and velocity (V_m) are not biased. It is given that the **measured** angle of attack (α_m,), angle of sideslip (β_m), and velocity (V_m) are related to their **true** values ($\alpha_{true}, \beta_{true}, V_{true}$) by the following formulas:

$$\alpha_m = \alpha_{true}(1 + C_{\alpha_{up}}) + v_\alpha$$

$$\beta_m = \beta_{true} + v_\beta$$

$$V_m = V_{true} + v_V$$

With $C_{\alpha_{up}}$ an unknown upwash coefficient that must be estimated, and with v_α, v_β, and v_V white noise sequences. The true angle of attack, angle of sideslip, and velocity are related to the system states as follows:

$$\alpha_{true} = \tan^{-1}\left(\frac{w}{u}\right)$$

$$\beta_{true} = \tan^{-1}\left(\frac{v}{\sqrt{u^2 + w^2}}\right)$$

$$V_{true} = \sqrt{u^2 + v^2 + w^2}$$

Substituting these expressions in the equations for α_m, β_m, and V_m will give you the observation equations $h(x,u,t) + \begin{bmatrix} v_\alpha & v_\beta & v_V \end{bmatrix}^T$.

You can use the following statistics for the process noise:

$$E\{\underline{w}(t_i)\} = 0; \quad E\left\{\underline{w}(t_i)\underline{w}^T(t_i)\right\} = Q\delta_{ij}; \qquad Q = diag(\sigma_{w_u}^2, \sigma_{w_v}^2, \sigma_{w_w}^2, \sigma_{w_C}^2)$$

$$\sigma_{w_u} = 1e-3, \sigma_{w_v} = 1\text{e}-3, \sigma_{w_w} = 1\text{e}-3, \sigma_{w_C} = 0$$

And the following sensor noise statistics:

$$E\{\underline{v}(t_i)\} = 0; \quad E\left\{\underline{v}(t_i)\underline{v}^T(t_j)\right\} = R\delta_{ij}; \quad R = diag\left(\sigma_{v_\alpha}^2, \sigma_{v_\beta}^2, \sigma_{v_V}^2\right)$$

$$\sigma_{v_\alpha} = 1.5e-3, \sigma_{v_\beta} = 1.5e-3, \sigma_{v_V} = 1.0e0$$

Make a smart choice for initializing your diagonal state prediction covariance matrix $P(0|0)$, as it will determine how fast your filter will converge depending on how close your initial guess for the state vector is to the real state vector.

Hint: Use a fixed time step of $\Delta t = 0.01$ in your (Runge-Kutta) integrator and 'c2d' operations!

Hint2: Note that the system dynamics equations $f(x,u,t)$ are trivial in the sense that they are fully described by the system input.

Appendix C: Description of Matlab Files

<F16traindata_CMabV_2024.mat>, <F16traindata_CMabV_2024.csv>
The F-16 aerodynamic dataset used in this year's multivariate spline assignment (CSV data for Python users).

<F16validationdata_2024.mat>, <F16validationdata_2024.csv>
The 'special validation dataset' used in this year's multivariate spline assignment, is used to calculate a final residual RMS value of your chosen models (CSV data for Python users).<load_f16data2024.m>
Use this script in Matlab to load the F-16 data from disk.

<F16_PlotData.m>
This script shows how to interpret and plot the F-16 data in the data file.

<expIOmapping.m>
This script demonstrates how to evaluate (using the <simNet.m> script) the example feed-forward and RBF neural networks using in the files <NetExampleFF.mat> and <NetExampleRBF.mat>.

<simNet.m>
This script calculates the output of feed-forward and RBF neural networks.

<NetExampleFF.mat>
Data file containing an example feed-forward neural network.

<NetExampleRBF.mat>
Data file containing an example RBF neural network.

Appendix D: Creating a neural network

In order to have comparable answers between students and to facilitate the readability of the scripts, a predefined network structure is given. The feedforward neural network is a structure with the following fields:

Variable fields

- .IW : input weights [# hidden neurons, # input neurons]
- .LW : output weights [# output neurons, # hidden neurons]
- .b{1,1} : input bias weights [# hidden neurons, 1]
- .b{2,1} : input bias weights [# output neurons, 1]
- .range : bounds on the input space [# input neurons, 2]
- .trainParam. : structure containing fields used during training. The fields are:
- .epochs : maximal number of epochs during training [1,1]
- .goal : desired performance, training stops is goal reached [1,1]
- .min_grad : minimal gradient, training stops if abs gradient value drops below the value (typically taken as 1e-10)
- .mu : learning rate, adapted during training [1,1]

Fixed fields

- .name{1,1} : network type = 'feedforward'
- .trainFunct{1,1} : act. Function hidden layer = 'tansig'
- .trainFunct{2,1} : act. Function output layer = 'purelin'
- .trainAlg{2,1} : training algorithm = 'trainlm'

As an example see NetExampleFF.mat.

For the RBF networks the structure has the same fields except that the bias fields are removed and the following fields are added/changed:

- .trainFunct{1,1} = 'radbas'. T
- .centers : center weights [# hidden neurons, # input neurons]
- .name{1,1} : network type = 'rbf'

As an example see NetExampoleRBF.mat. If all fields are correctly filled than the supplied function simNet.m can be used to simulate the network output for a given set of inputs. To show how the function simNet.m works are how the IO mapping can be represented, an example is provided: see expIOmapping.m.