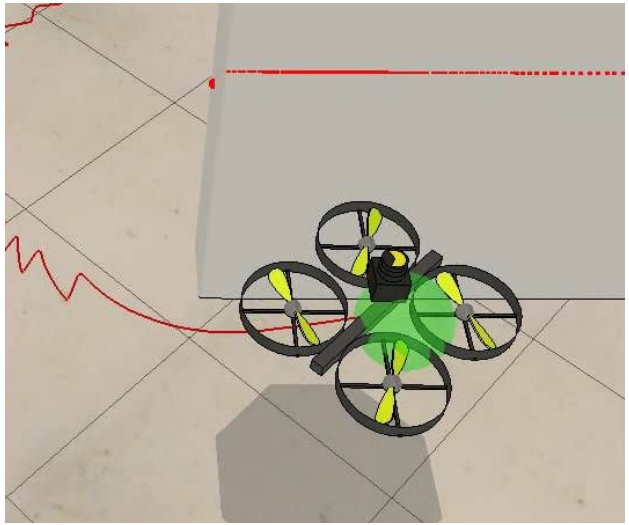
System Identification

CS6744 ML for Robotics - Groups 1 and 4

Problem Overview

Find the system model that best represents a simulated drone

1. Data collection
2. Non-linear regression
3. Evaluation



Step 1: Data collection

- Representative dataset
- Careful with sampling rates!

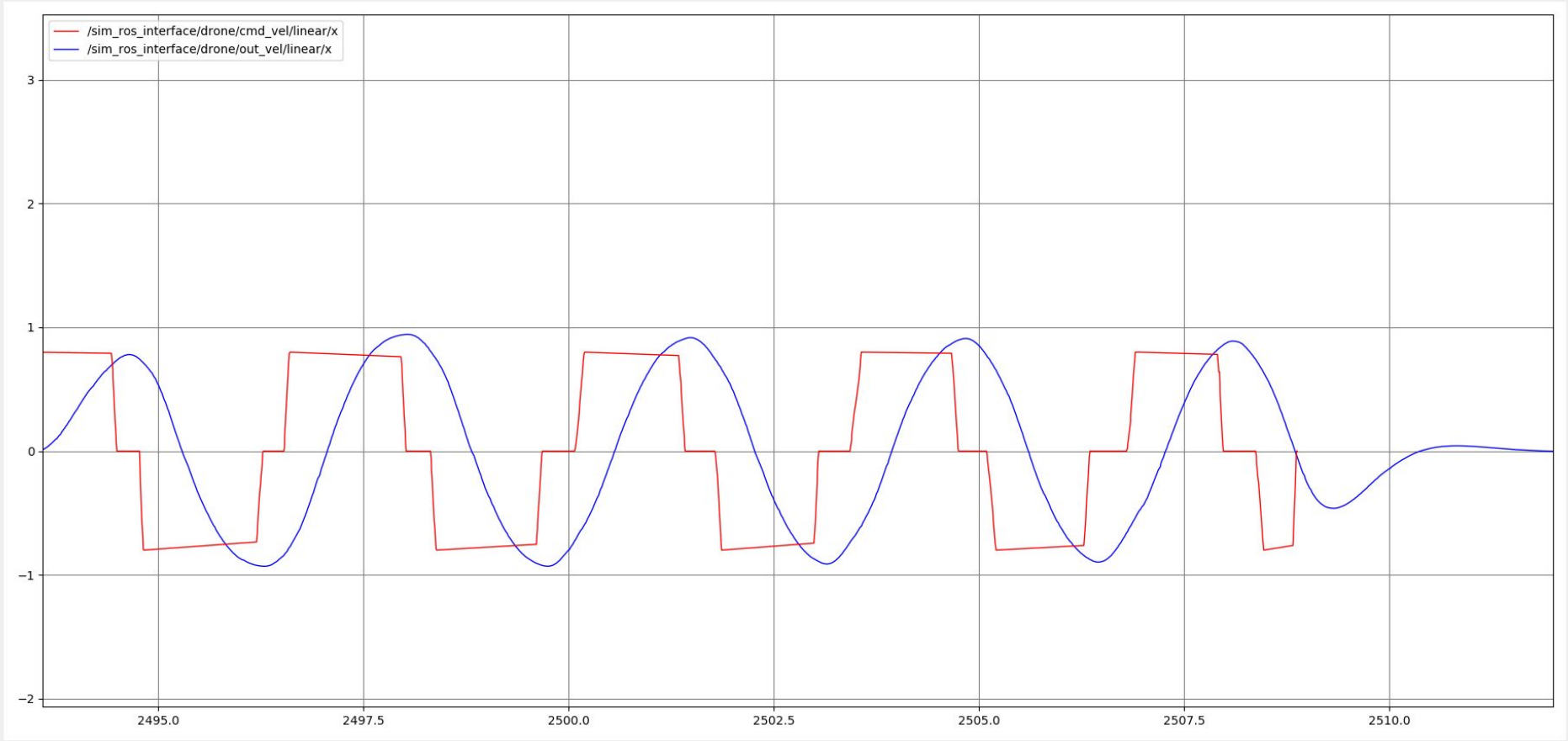
rqt_plot__Plot - rqt

MatPlot

Topic /predictor/prediction

autoscroll

x=2504.48 y=1.94695



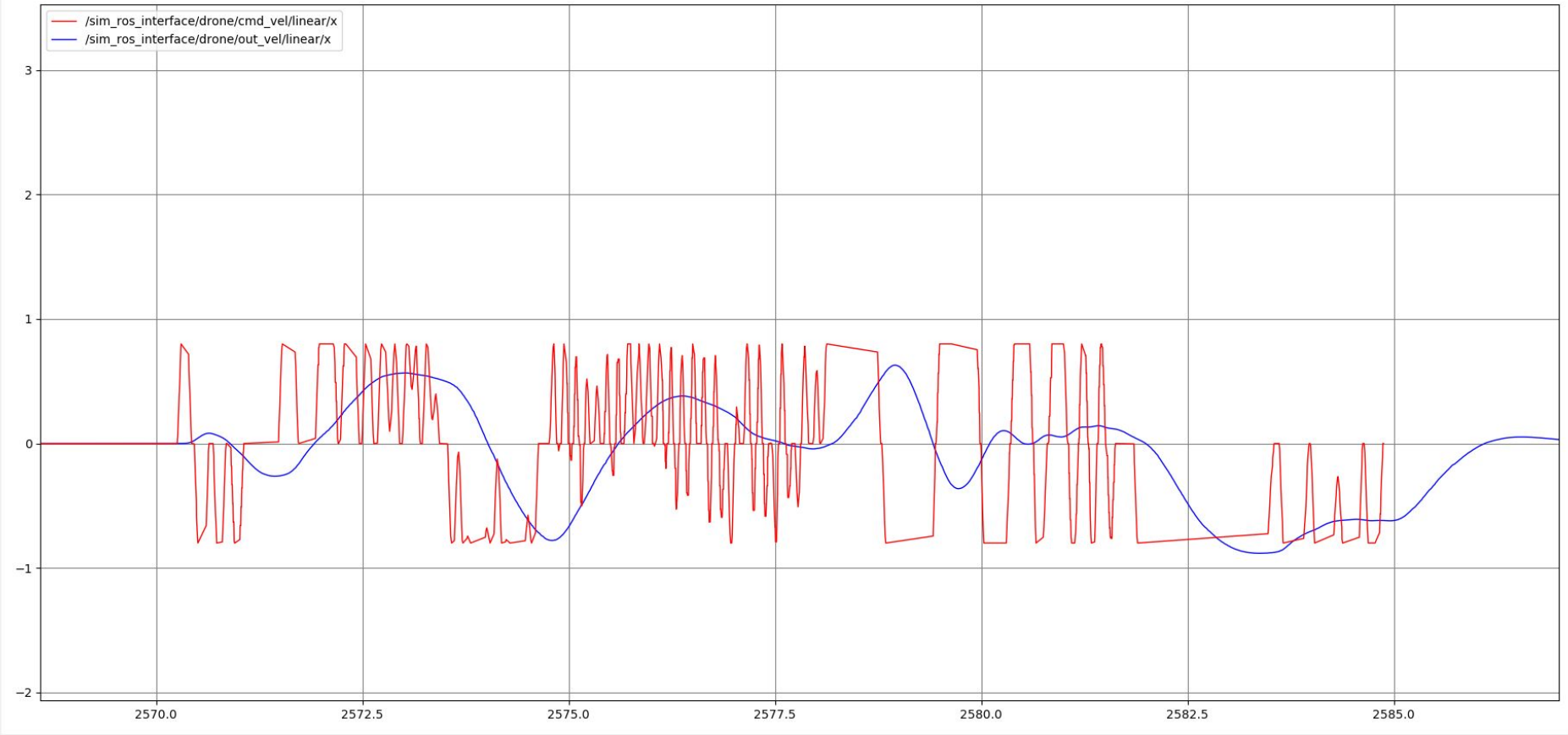
rqt_plot__Plot - rqt

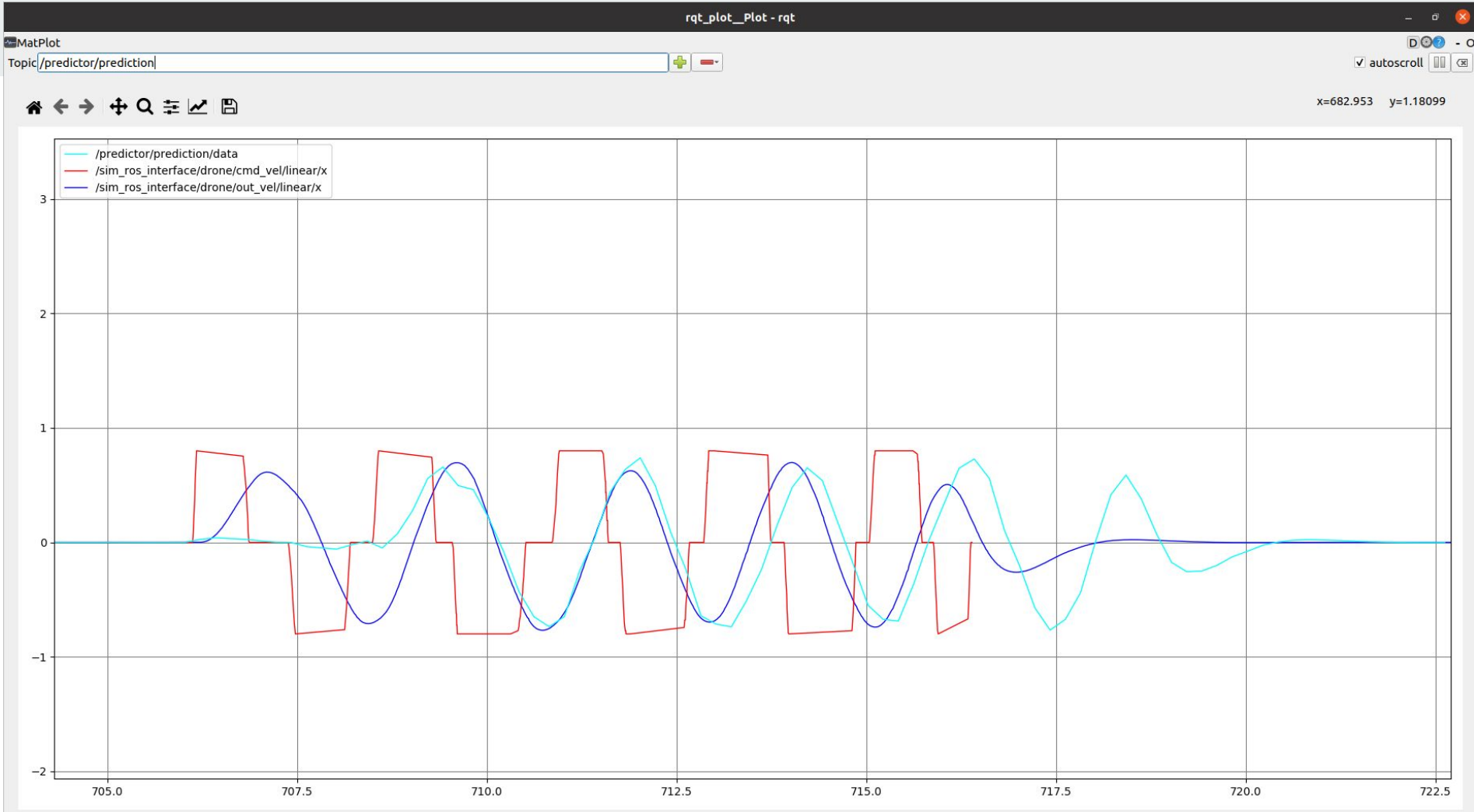
MatPlot

Topic /predictor/prediction

autoscroll

x=2579.56 y=1.87731




rqt_plot__Plot - rqt
MatPlot
Topic /predictor/prediction
autoscroll
x=682.953 y=1.18099
/predictor/prediction/data
/sim_ros_interface/drone/cmd_vel/linear/x
/sim_ros_interface/drone/out_vel/linear/x
3
2
1
0
−1
−2
705.0
707.5
710.0
712.5
715.0
717.5
720.0
722.5

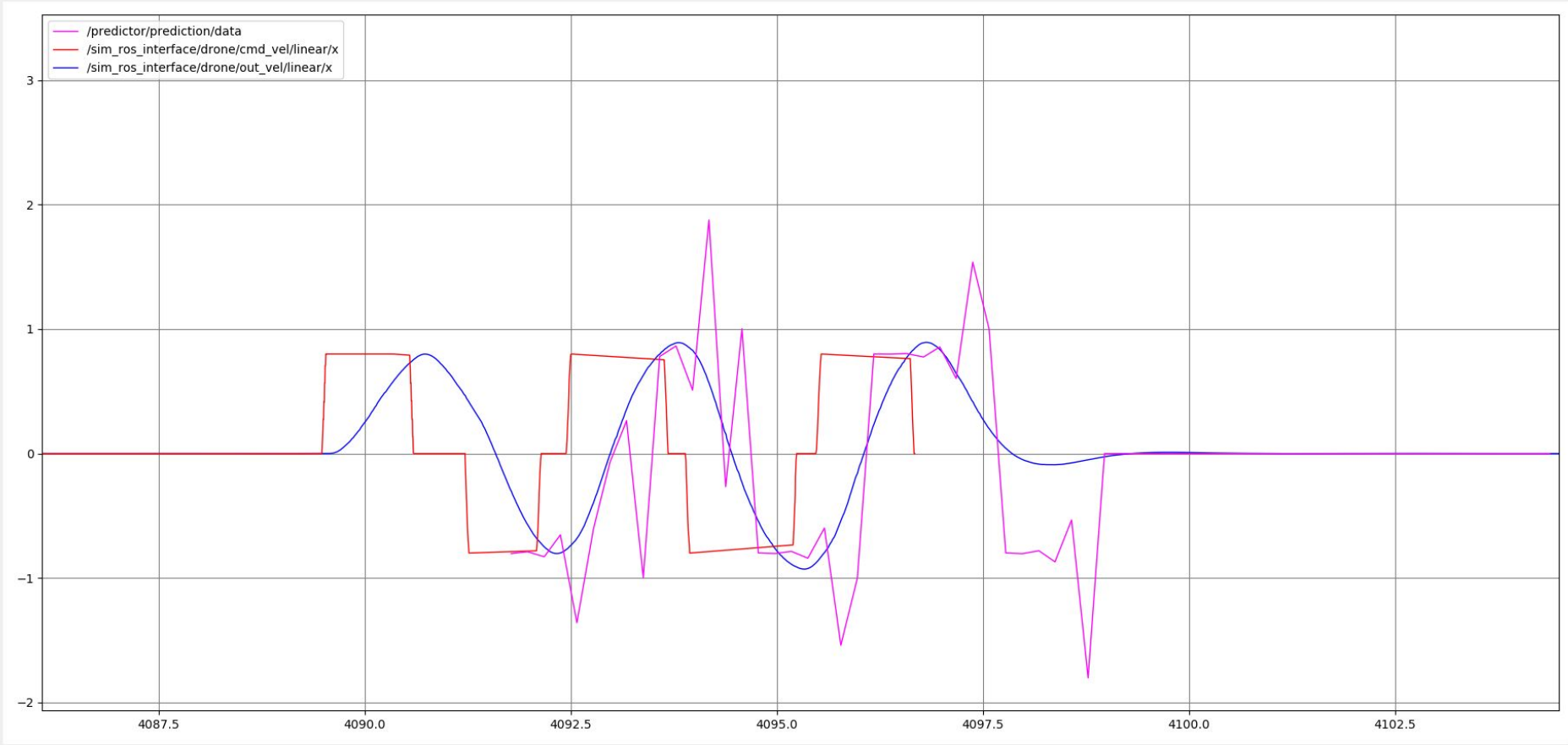
rqt_plot__Plot - rqt

MatPlot

Topic /predictor/prediction

autoscroll

x=4096.7 y=2.09317



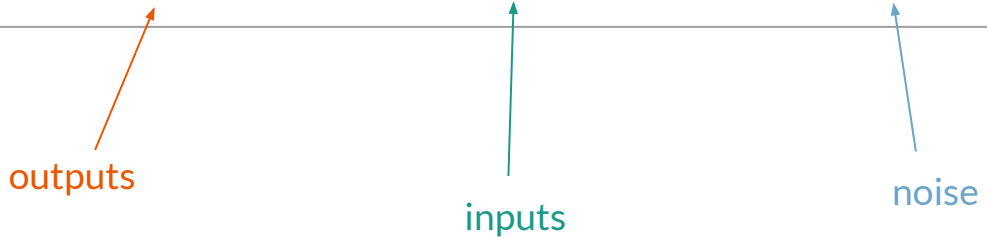
Step 2: Non-linear regression

- Make models
- Grid search!
- Throw in some regularization

Models

Base	$A(q)y(t) = B(q)u(t) + C(q)e(t)$

Each parameter varies the model's dependence on…



Models

Base	$A(q)y(t) = B(q)u(t) + C(q)e(t)$
ARX	$y(t) + a_1 y(t-1) + \dots + a_{na} y(t-n_a) = b_1 u(t) + \dots + b_{nb} u(t-n_b) + e(t)$

No dependence on previous noise

Models

Base	$A(q)y(t) = B(q)u(t) + C(q)e(t)$
ARX	$y(t) + a_1 y(t-1) + \dots + a_{na} y(t-n_a) = b_1 u(t) + \dots + b_{nb} u(t-n_b) + e(t)$
FIR	$y(t) = b_1 u(t) + \dots + b_{nb} u(t-n_b) + e(t)$

No dependence on previous noise or outputs

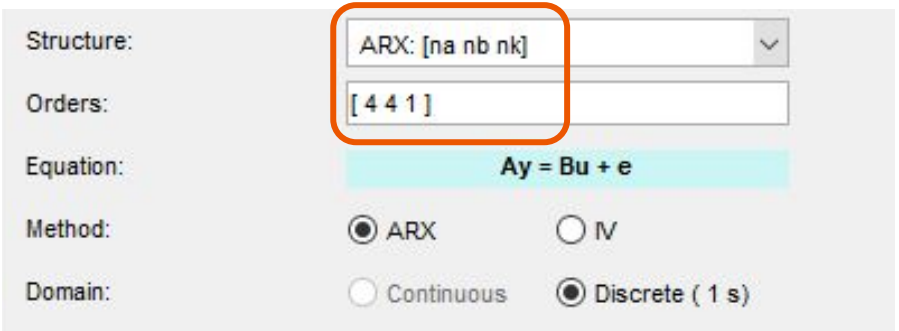
Goals

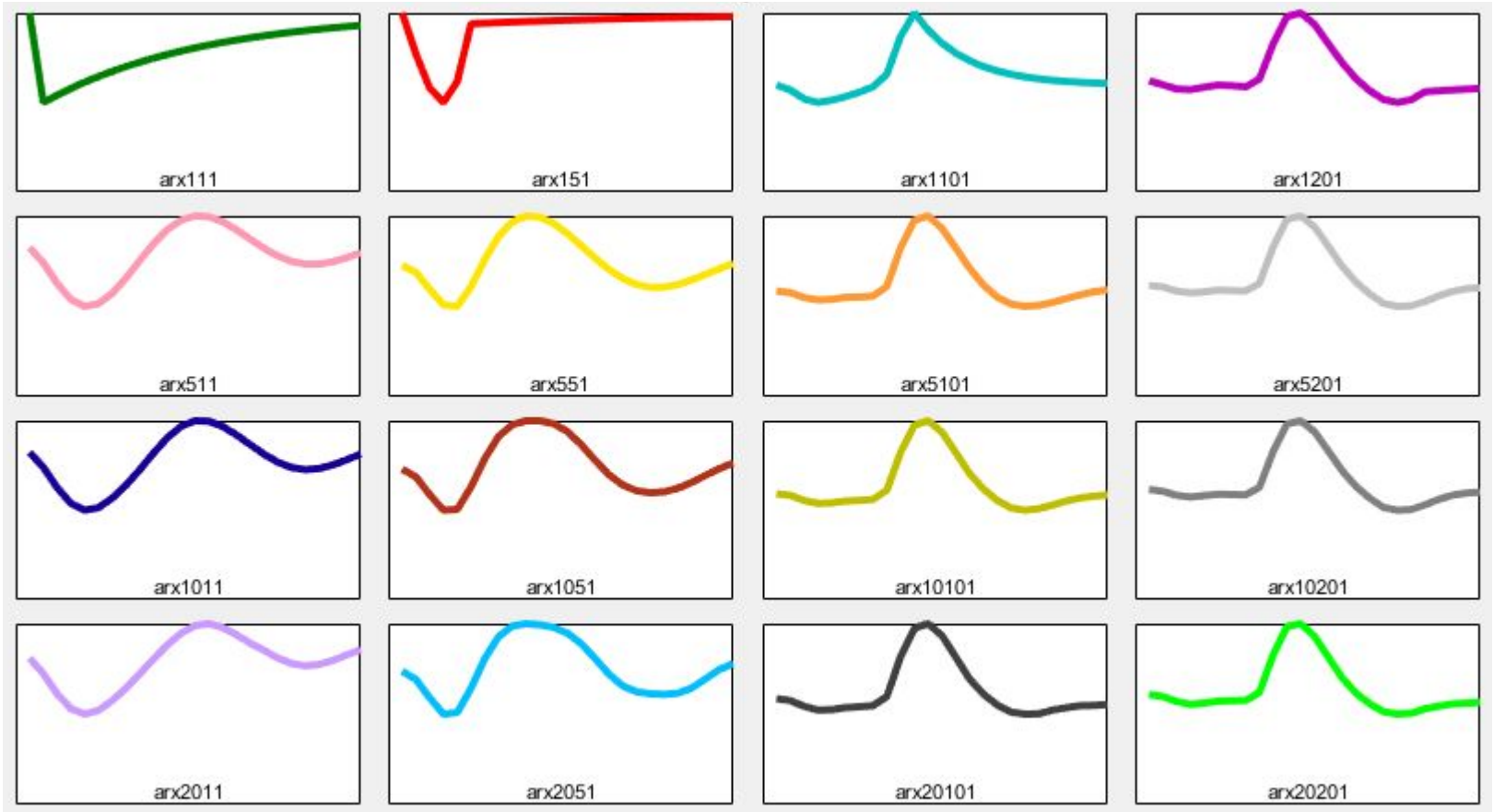
Determine the system's dependence on previous outputs, inputs, and noise

Search for parameters (a_1 ... a_{na}) and (b_1 ... b_{nb}) that minimize estimation error

Matlab

Make ARX models varying the orders of `A` and `B` (`na`, `nb`),
and no input-output delay (`nk=1`)




arx111
arx151
arx1101
arx1201
arx511
arx551
arx5101
arx5201
arx1011
arx1051
arx10101
arx10201
arx2011
arx2051
arx20101
arx20201

Grid Search

Search space:

- `na = 0, 1, 2, … , 20`
- `nb = 0, 1, 2, … , 20`
- `nk = 1`

Note:

- when `na = 0`, we have an FIR model
- `step` is how far out you want to predict the output

```
model = arx(train_data, [na nb nk]) % train a model

validation = compare(validation_data, model, step) % validate the model
```

Results

441 models created

20-step prediction

Compared with validation data for accuracy

272 models with > 96 % accuracy

Best 10 models: all ARX

accuracy	orders		
96.521	16	0	1
96.519	15	0	1
96.518	18	0	1
96.517	17	0	1
96.515	14	0	1
96.509	19	0	1
96.503	20	0	1
96.475	16	2	1
96.474	16	1	1
96.474	16	3	1

Worst 10 models: all FIR

accuracy	orders		
-0.32525	0	0	1
15.974	0	20	1
16.06	0	19	1
16.151	0	18	1
16.244	0	17	1
16.34	0	16	1
16.438	0	15	1
16.541	0	14	1
16.646	0	13	1
16.756	0	12	1

Results

Including the best, worst, and in between

Best Fits

arx1601: 99.98

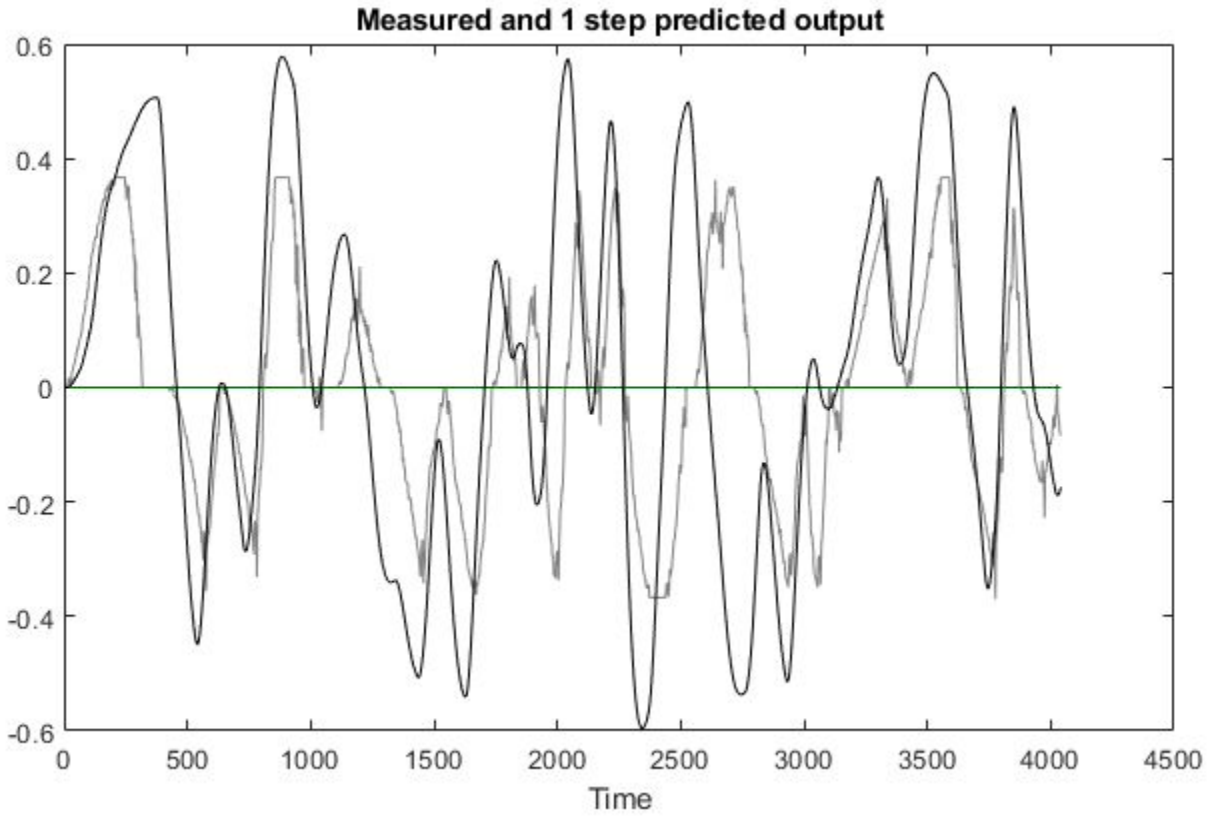
arx1081: 99.98

arx401: 99.98

arx201: 99.95

fir0121: 16.76

fir001: -0.3253



Results

5-step prediction

Best Fits

arx1601: 99.86

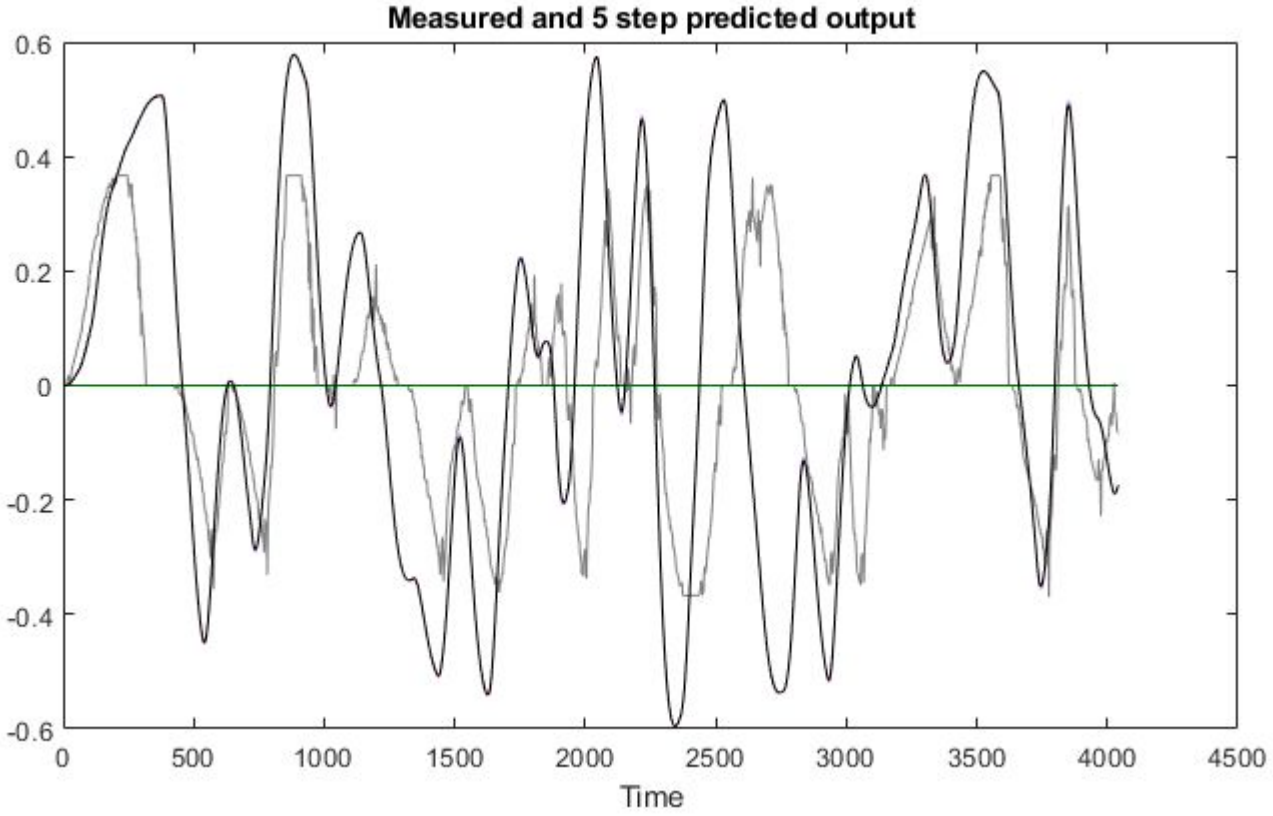
arx1081: 99.86

arx401: 99.81

arx201: 99.34

fir0121: 16.76

fir001: -0.3253



Results

20-step prediction

Best Fits

arx1601: 96.52

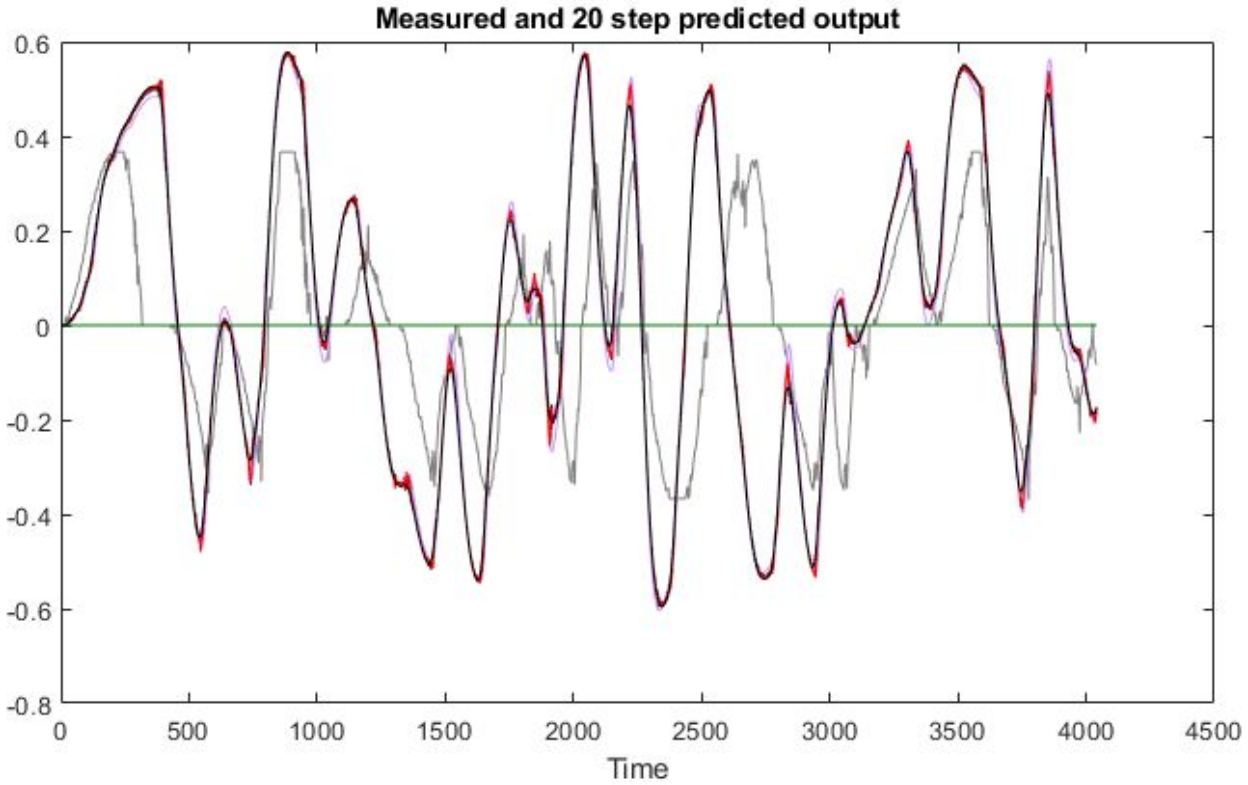
arx1081: 96.37

arx401: 96.02

arx201: 91.32

fir0121: 16.76

fir001: -0.3253



Regularization

Number of parameters seems really high for the physics of the system

Tried regularization

Had barely any effect on these results

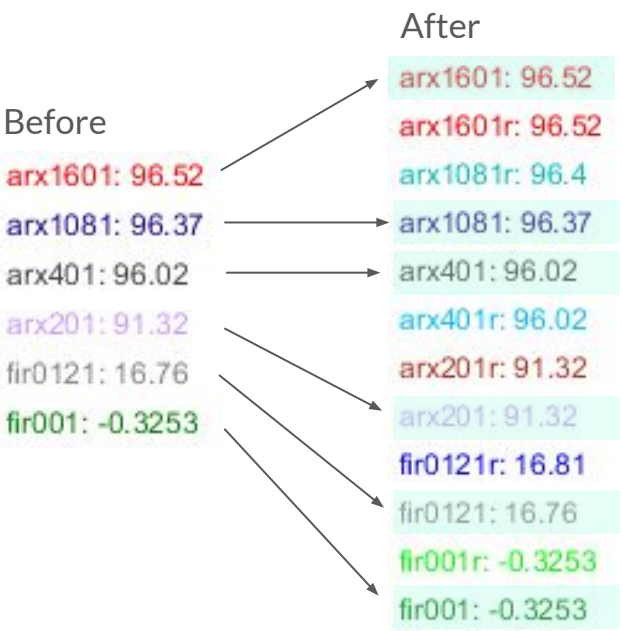
See Later:

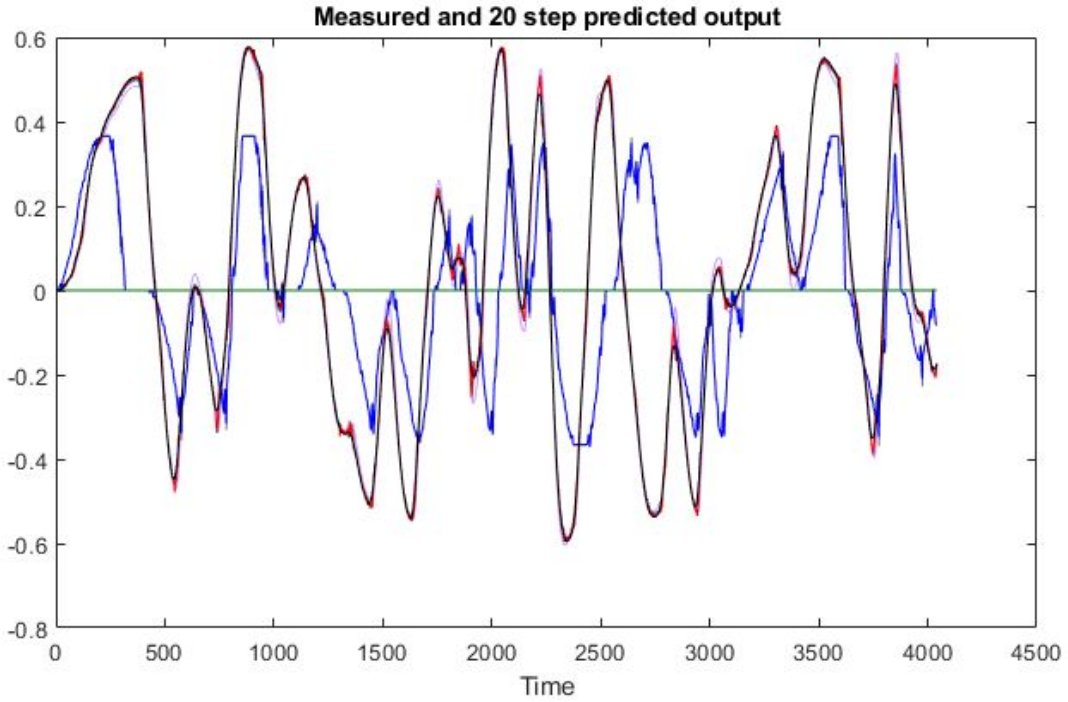
Regularization helps in the simulation

Best 10 models: basically all same

accuracy	orders		
96.52	16	0	1
96.518	15	0	1
96.518	17	0	1
96.518	18	0	1
96.514	14	0	1
96.51	19	0	1
96.504	20	0	1
96.491	18	7	1
96.49	18	9	1
96.489	17	12	1

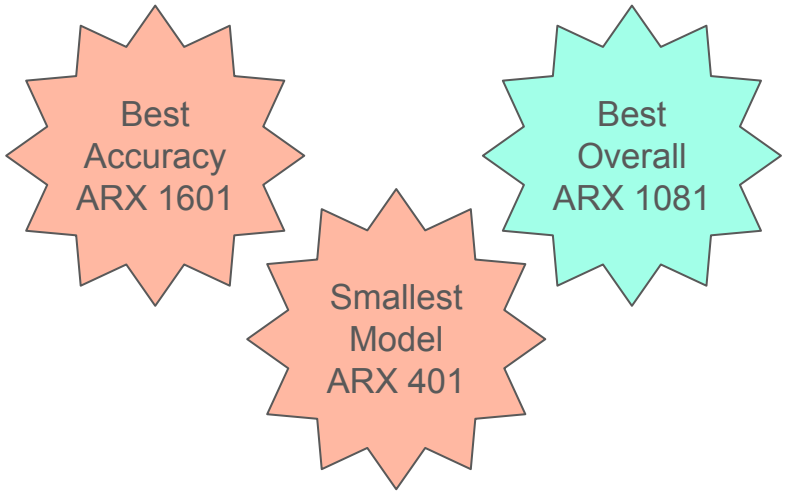
Regularization





Best Models

Determined by human judgement

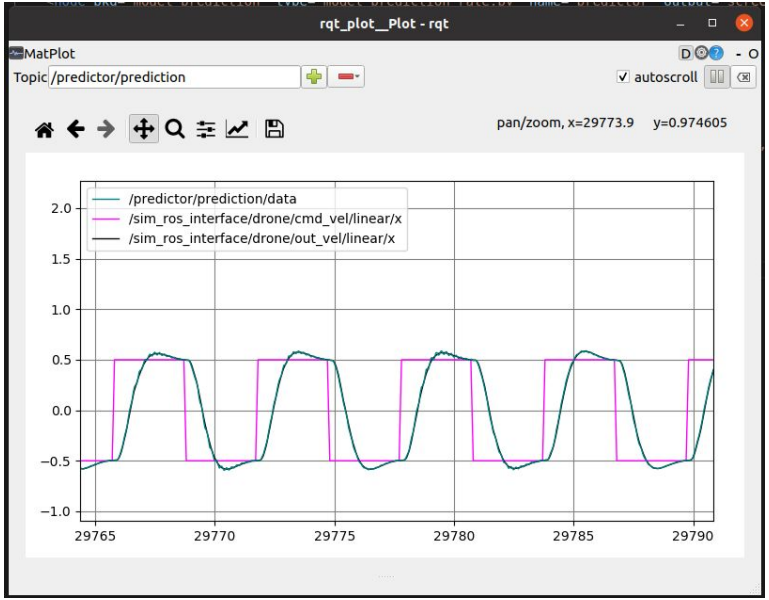


Model	Accuracy	Regularization Accuracy
ARX 1601	96.5210 %	96.5204 %
ARX 401	96.0165 %	96.0161 %
ARX 1081	96.3723 %	96.3982 %

Step 3: Evaluation

- Let’s have fun!
- 1, 5, 20 step ahead prediction

1-step ahead prediction



using ARX [10 8 1] model

command_coef_csv = ”0.00008586, -0.00019776, 0.00027644, -0.00018129, 0.00019877, -0.00023495, 0.00011212, -0.000049284"
State_coef_csv = "-1.4631, -0.5476, 0.5789, 0.5447, 0.5040, -0.1669, -0.2962, -0.4064, 0.0790, 0.1737"

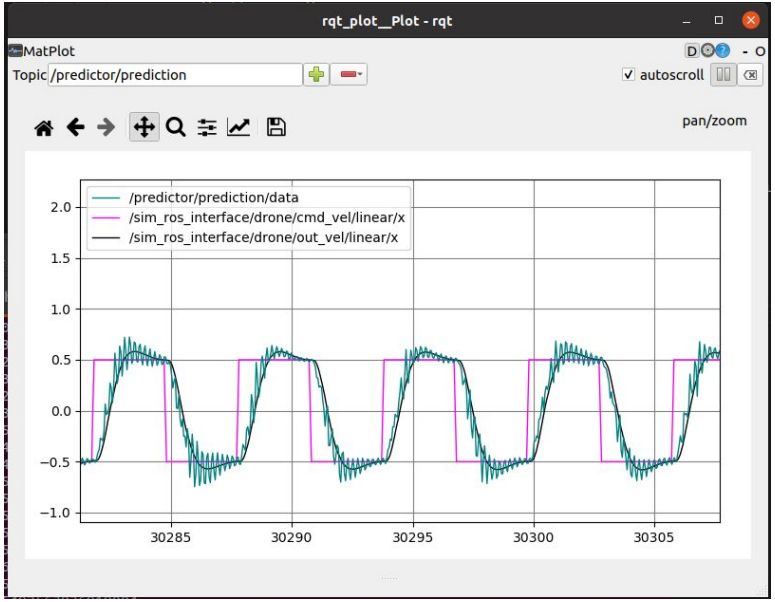
Almost all models perform well in 1-step ahead prediction

For N-steps ahead prediction

```
x1 = x
u1 = u
for i in range (0,1):
    pred = sum([-a*xi for a,xi in zip(self.state_coef,x1)]) + sum([b*ui for b,ui in zip(self.command_coef,u)])
    x1.append(pred)
    u1.append(u1[-1])
    x1 = x1[-len(self.state_coef):]
    u1 = u1[-len(self.command_coef):]

print(pred)
```

5-step ahead prediction



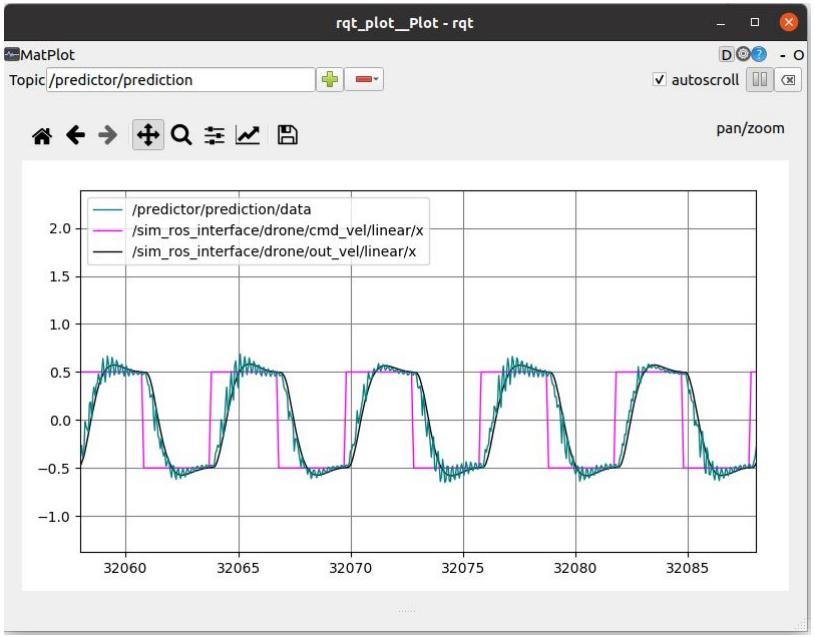
Still using ARX [10 8 1] model

The result is very noisy this time

Possible reasons:

- ARX model parameters not good enough
- Sampling frequency is too high(80Hz currently)

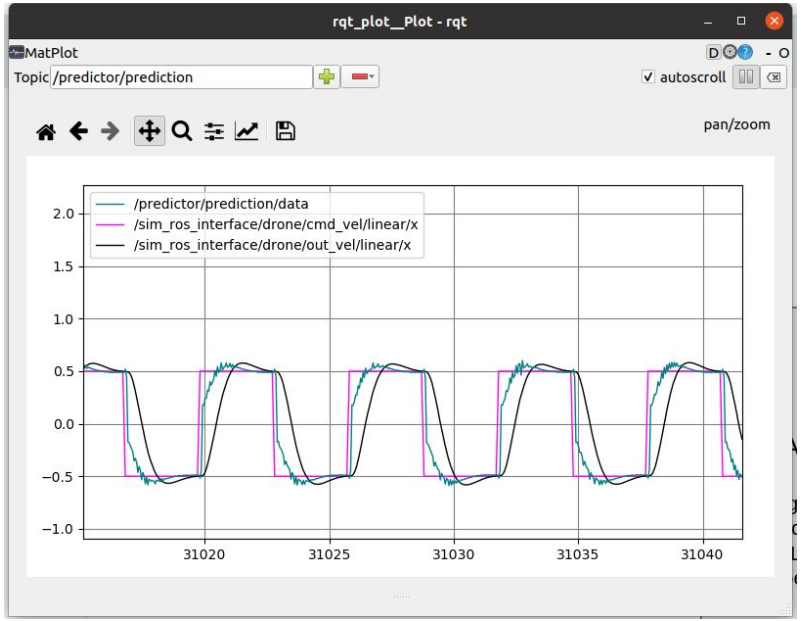
5-step ahead prediction



using ARX [10 8 1] model with regularization

Regularization make the model more stable

20-step prediction ahead



using ARX [4 4 1] model credit: Devarsi

Sampling at 30Hz
command_coef_csv = “-0.00040267, 0.0013, 0.00019189, 0.0087”
State_coef_csv = “-1.6873, 0.8064, -0.3918, 0.2823”

The model is very robust

Thank you!